UNIVERSIDAD POLITÉCNICA DE CARTAGENA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES

DEPARTAMENTO DE TECNOLOGIA ELECTRÓNICA

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES (G.I.T.I.)

"DISEÑO Y CONSTRUCCIÓN DE UN CUADRICÓPTERO DE BAJO COSTE CONTROLADO POR UN DISPOSITIVO MOVIL INTELIGENTE"

DIRECTORES: D. Juan Antonio López Riquelme

D. Andrés Iborra García

AUTOR: Enrique González Sancho

OCTUBRE 2014

AGRADECIMIENTOS

- A JUAN ANTONIO POR SU DISPONIBILIDAD Y SU AYUDA
- A MI PADRE, POR TRANSMITIRME SU PASIÓN POR APRENDER
- A MI TÍO, POR PERMITIRME MOLESTARLE CON LA IMPRESIÓN 3D
- A MI FAMILIA, POR ESTAR SIEMPRE AHÍ
- A MIS AMIGOS, POR RECORDARME QUE DESCANSAR TAMBIÉN ES IMPORTANTE
- A TODOS LOS DEMÁS

Índice

Capítulo 1

Introducción

1.1 Introducción

Cada día hay más proyectos sobre drones no tripulados, pequeñas aeronaves que combinan el control automático con el manual, para diferentes aplicaciones. En concreto, los cuadricópteros tienen ya una presencia notable, debido a su relativa simplicidad, y a las numerosas posibilidades que ofrecen, desde la fotografía y la filmación aérea hasta las misiones de reconocimiento y de rescate. Existe ya una gran variedad de diseños, diferentes tamaños y potencias, según la aplicación. También hay varias configuraciones posibles, podemos encontrar tricópteros, cuadricópteros, hexacópteros, decacópteros, etc.

Los más conocidos, son los que vemos ya en algunas noticias, generalmente utilizados para la fotografía y la filmación aérea. En lo relativo a esta aplicación, los drones-cámara se pueden considerar como el siguiente nivel de las cámaras tipo Go-Pro. Si ya estas cámaras se utilizan para realizar vídeos de deportes de riesgo, en los que no es fácil portar una cámara normal, los drones añaden a esta función la ventaja de un ángulo de visión completamente libre.

Algunas empresas los utilizan para el reparto de los pedidos a domicilio. Es el caso de Amazon o Google, que utilizan diferentes modelos de multi-cópteros para este fin.

También están siendo muy utilizado a nivel de investigación en diversos casos de estudio: seguridad, medida de parámetros en cultivos, montaje de estructuras, etc.

Todas las aplicaciones que puedan tener, resultan muy llamativas debido a su carácter innovador y a su rendimiento.

1.2 Objetivos

Los objetivos de este proyecto son los siguientes:

- Recopilar información sobre los modelos actuales y sus características.
- Seleccionar los componentes óptimos para diseñar y construir uno propio.
- Integrar las diferentes partes del hardware elegido.
- Diseñar el código que hará funcionar correctamente al cuadricóptero.
- Realizar las calibraciones y pruebas necesarias para su uso.
- Controlar el dron desde un Smartphone por medio de una aplicación móvil.

1.3 Descripción de la memoria

La presente memoria consta de los siguientes capítulos:

Capítulo 1 – Introducción

En este primer capítulo se presenta el trabajo y sus objetivos, y se describen los capítulos que componen la memoria.

Capítulo 2 – Estado del arte

En el Capítulo 2 se presentan los modelos más importantes que existen en el campo de los drones, se enumeran sus características y se hace una pre-selección de los componentes a utilizar.

Capítulo 3 – Materiales y métodos

El capítulo 3 describe detalladamente cada elemento seleccionado, justificando teórica y experimentalmente, cuando es necesario, la selección y el ajuste realizados en cada caso.

Capítulo 4 – Diseño y montaje del hardware

En el capítulo 4 se explica el diseño del cuadricóptero, a partir de los elementos seleccionados en el capítulo anterior y se narra el montaje y la problemática asociada al mismo.

Capítulo 5 – Desarrollo del software de vuelo y calibración

En este capítulo se explica el software de vuelo desarrollado, incluyendo los conceptos teóricos necesarios para la comprensión del mismo, y los ajustes de los parámetros de estabilidad del drone.

Capítulo 6 – Conclusiones y trabajos futuros

En el último capítulo se reúnen las conclusiones obtenidas tras la realización del trabajo, y se comentan las tareas y trabajos relacionados que se salen de los objetivos de éste, pero que se desean realizar en el futuro.

Anexo I – Código

En este anexo se incluye el código del software de vuelo. En el Capítulo 5 se muestran sólo fragmentos importantes, para una mejor comprensión de su funcionamiento, pero es aquí donde se recopila en su totalidad.

Capítulo 2

Estado del Arte

2.1 Introducción

Hay diferentes formas de hacerse con un dron o cuadricóptero: se puede comprar directamente un modelo comercial, o puede construirse un modelo personalizado. Los modelos comerciales garantizan un vuelo estable, tras una pequeña calibración que debe realizar el usuario. También suelen incorporar elementos adicionales como cámaras o sensores para funciones específicas.

2.2 Modelos comerciales

Se pueden encontrar multitud de modelos comerciales de diferentes tamaños y características, que vienen en kits preparados para montar y volar. Los modelos más destacados son:

2.2.1 Parrot AR.Drone 2.0

El AR.Drone 2.0 es un cuadricóptero comercial de 77,7 x 38,3 x 12,5mm con GPS integrado. La empresa ofrece diferentes modelos y accesorios: cámaras HD, registradores de vuelo con autopiloto para volver al punto de despegue, mandos de control remoto y aplicaciones para Smartphone con las que controlar el dron. Concretamente, en la Tabla 2.1 se presentan algunas de las especificaciones técnicas de este modelo:

Grabación de video en HD: • Cámara HD 720p 30 FPS. • Lente gran angular: Diagonal 92°. • Perfil base de codificación H264. • Transmisión de baja latencia. • Almacenamiento de vídeos durante el vuelo. • Foto JPEG. • Almacenamiento instantáneo de vídeos con Wi-Fi.
Estructura: • Tubos de fibra de carbono. • Peso total 380 g. con carcasa de exterior, 420 g con carcasa de interior. • Fibra de alta calidad (30%) cargada con piezas plásticas de nylon. • Espuma para aislar el centro inercial de las vibraciones del motor. • Casco EPP inyectado por un molde de metal sinterizado. • Nano-revestimiento que repele los líquidos en los sensores de ultrasonidos. • Completamente reparable: Todas las piezas e instrucciones para la reparación están disponibles en Internet.
Asistencia electrónica: • Procesador de 1 GHz y 32 bits ARM Cortex A8 con vídeo DSP TMS320DMC64x de 800 MHz. • Linux 2.6.32. • RAM DDR2 de 1GB a 200MHz. • USB 2.0 de alta velocidad para extensiones • Wi-Fi b/g. • Giroscopio de 3 ejes con una precisión de 2000°/seg. • Acelerómetro de 3 ejes con una precisión de +/- 50mg. • Magnetómetro de 3 ejes con una precisión de 6°. • Sensor de presión con una precisión de +/- 10 Pa. • Sensores de ultrasonido para medir la altitud de avance. • Cámara vertical QVGA de 60 FPS para medir la velocidad de avance.
Motores: • 4 motores "inrunner" sin escobillas de14,5W y 28500 RPM. • 1 Cojinete de microbola • 1 Engranajes de Nylatron de bajo ruido para 1 reductor de propulsión 8,75. • 1 Eje de transmisión de acero templado. • 1 Cojinete de bronce auto-lubricante. • 1 Resistencia aerodinámica específica de alta propulsión para ofrecer una excelente maniobrabilidad. • 1 CPU AVR de 8 MIPS por controlador de motor. • Parada de emergencia controlada por software. • Controlador de motor totalmente reprogramable. • Controlador electrónico del motor resistente al agua.

Tabla 2.1.- Características del AR.Drone.

Figura 2-1.- Vista del AR.Drone.

2.2.2 *DJI Phantom*

La empresa DJI ofrece un cuadricóptero destinado a la filmación aérea, con las siguientes especificaciones:

- Envergadura: 350 mm.
- Diseño atractivo e integrado.
- Unidad de control remoto (4 pilas AA).
- Estable, ágil y fácil de volar.
- Dos modos de control de vuelo, con "Position hold".
- Control de orientación inteligente (IOC).
- Aterrizaje a prueba de fallos y función "*auto go home*".
- Protección frente a bajo voltaje.
- Velocidad máxima de vuelo: 10 ó 15 m/s (según modelo).
- LEDs de alta intensidad, para visualizar la velocidad de vuelo.
- Batería de larga duración (5200 mAh en la mayoría de modelos).

Existen diferentes modelos, con diferentes cámaras (1080p ó 720p), con diferentes alcances en la descarga de video (700m ó 300m), y de control remoto (500, 800 y 1000m).

Figura 2-2.- Vista del dron DJI Phantom.

2.3 Modelos no comerciales

A continuación se exponen diferentes modelos desarrollados por universidades y por aficionados. De los proyectos de universidad se dispone de poca información técnica, mientras que de los "caseros" hay guías detalladas de construcción y configuración. La información proporcionada por los proyectos "caseros" es muy valiosa, ya que sirven de guía a la hora de seleccionar los componentes. No obstante, esto debe utilizarse sólo como referencia, ya que la configuración óptima depende de la aplicación y de las características que se desean para cada proyecto.

2.3.1 Pelican

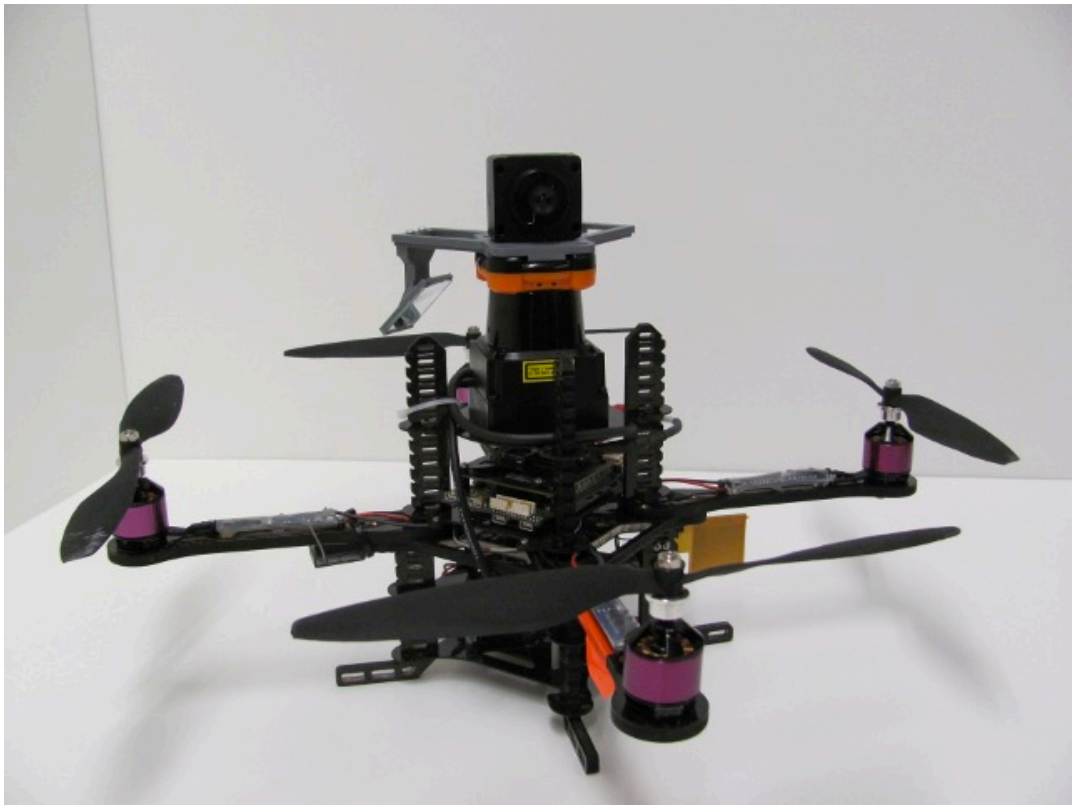


Figura 2-3.- Cuadricóptero modelo Pelican de AscTec.

Desarrollado por CCNY Robotic Lab en colaboración con AscTec. El Pelican [1] pesa 630 g y es capaz de cargar con 650 g adicionales. Tiene herramientas de programación y configuración propias (AscTec SDK y AscTec Simulink Toolkit). Está equipado con cámara y escáner láser para realizar funciones de movimiento avanzadas y secuencias de reconocimiento del entorno.

Se trata de una plataforma robótica aérea muy utilizada en proyectos de investigación y tiene un coste considerablemente elevado de unos 4000 $.

2.3.2 Penn quadrotors

Figura 2-4.- Fotografía de una prueba realizada en el proyecto Penn quadrotors.

Es un proyecto desarrollado por la universidad de Pennsylvania [2]. Está especialmente dirigido a la cooperación de nano-quads. Éstos están normalmente controlados por un ordenador externo, que hace todo el cálculo de posiciones relativas entre los quads, haciendo formaciones, e incluso llevando a cabo tareas de cooperación como la construcción de pequeñas estructuras, tal y como se puede ver en [3].

2.3.3 PixHawk Cheetah Bravo

Figura 2-5.- Fotografía del dron PixHawk Cheetah Bravo.

La ETH de Zurich ha realizado un quad equipado con un sistema de cámaras estéreo para la navegación en interiores. En [4] se puede encontrar más información.

2.3.4 Controlador ROS para el Parrot AR. Drone

La universidad Simon Fraser de Canadá ha implementado un driver [5] para un quad comercial (Parrot AR. Drone y AR. Drone 2.0), basado en el middleware ROS (*Robotic Operating System*). ROS es un conjunto de librerías software y herramientas que facilitan el diseño de aplicaciones con robots, tanto aéreos como terrestres.

2.3.5 Proyecto STARMAC

Figura 2-6.- Dron construido en el proyecto STARMAC.

La universidad de Stanford ha desarrollado un conjunto de vehículos autónomos, preparados para volar tanto en interiores como en exteriores, llevando diversas cargas y realizando tareas cooperativas [6]. El modelo de la Figura 2-6 tiene aproximadamente 120 cm de envergadura. Llevan incorporados protectores para las palas, para la navegación en interiores, y ofrecen un diseño robusto e integrado.

2.3.6 Drone de la MSU

Figura 2-7.- Dron desarrollado en la Universidad de Michigan.

La Universidad de Michigan ha trabajado en un quad que realiza medidas por medio diferentes sensores (radiómetro de alta resolución, cámara térmica, escáner láser)

para el cuidado de los cultivos en zonas agrícolas [7]. El dron sobrevuela una zona de cultivo, recogiendo datos sobre el suelo y las condiciones meteorológicas, para establecer qué zonas necesitan qué tratamientos para su crecimiento óptimo, minimizando los costes al no aplicar, por ejemplo, ciertos fertilizantes en las zonas en las que no es necesario.

2.3.7 *Aerial Constructions, ETH Zurich*

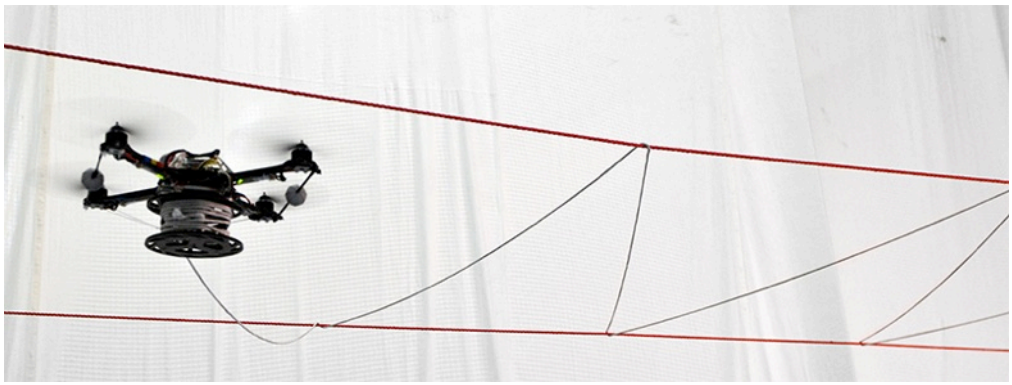


Figura 2-8.- Dron construido en la Universidad de Zúrich.

En la universidad de Zúrich están trabajando en diversos proyectos sobre drones [8]. Uno de ellos consiste en la utilización de quads para construir estructuras como la que se muestra en la Figura 2-8.

2.3.8 *Cornell University quad*

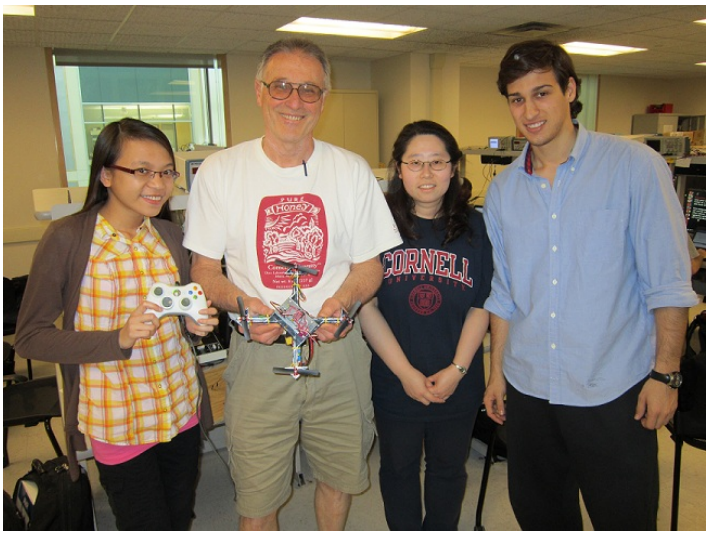


Figura 2-9.- Fotografía de un dron construido por estudiantes de la Cornell University.

En la universidad de Cornell un grupo de estudiantes ha diseñado un cuadricóptero de pequeño tamaño (unos 40 cm aproximadamente), con unos motores potentes en relación a su peso, lo que supone un gran potencial en cuanto a velocidad y maniobrabilidad [9].

2.3.9 Modelos hechos por aficionados

Como se ha comentado antes, los desarrolladores de los modelos que se muestran a continuación proporcionan mucha información técnica e incluso guías detalladas de montaje. Al igual que ocurre con los otros modelos, hay una gran variedad en los sensores utilizados y las características, dependiendo de la aplicación. Además, los aficionados que comparten sus proyectos en Internet, suelen hacerlo en foros en los que se puede encontrar asistencia de personas experimentadas, para solucionar rápidamente problemas por los que otros ya han pasado.

A continuación se presentan algunos de estos proyectos, junto con información variada y presupuestos desglosados.

2.3.9.1 MultiWii

Figura 2-10.- Fotografía del proyecto MultiWii.

EL proyecto MultiWii utiliza componentes de los controles de la videoconsola Wii. Este es un proyecto de código abierto en el que, a partir de los componentes especificados, se puede calcular el siguiente presupuesto:

- Arduino Mini Pro: 7,21 €
- 3 x Hobbycity Turnigy 3020 Brushless Outrunner Motor 1200kv = 36,24 €
- 3 x Hobbycity Hobbyking SS Series 8-10A ESC = 14,13 €
- Hobbycity Turnigy High Density R/C LED Flexible Strip-Green = 2,92 €
- 3 x Hobbycity GWS EP Propeller (DD-7035 178x89mm) = 9,57 €
- Hobbycity Turnigy MG90S Metal Gear Servo 1.8kg = 3,61 €

- Hobbycity Turnigy 1300 mAh 3S 25C Lipo Pack = 7,24 €
- Wii Motion Plus (Acelerómetro): 15,99 €
- Wii Nunchuck (Giróscopo): 2,95 €

Coste total: 99,86 €

Estimando un 20 % de otros costes (chasis, cables, costes indirectos, etc.), quedando un coste final de **119,84 €**.

2.3.9.2 ArduCopter

Figura 2-11.- Fotografía del modelo ArduCopter.

Los componentes principales del ArduCopter son:

- ArduPilot Mega: 49,99 €
- IMU Shield: 152,4 €
- Magnetómetro de 3 ejes HMC5883L: 29,90 €
- Servo cables: 6 x 4,99 € = 29,94 €
- ADAFRUIT GPS MTK3339: 31,10 €
- XBEE PRO 60 mW CON ANTENA: 32,70 €
- Motores Turnigy 2209 28turn 1050kV 15A outrunner: 4 x 8,66 € = 34,64 €

La cámara es opcional, y no se tendrá en cuenta para este proyecto.

Con los datos anteriores se obtiene un coste total de 360,67 €. Suponiendo un margen del 10% para gastos menores, habría que considerar un presupuesto de 400 €. Es un precio bastante más elevado que el del proyecto anterior, ya que cuenta con mayor cantidad de sensores, y además de mayor calidad. No obstante, el precio puede reducirse bastante cambiando algunos componentes, como una IMU más barata, la cual no es difícil de encontrar, o quitando algún sensor.

2.3.9.3 *Cuadricóptero casero:*

Fuente: http://blog.deinventos.com/construyendo-un-cuadricoptero-i/

Tamaño: 78 x 78 cm (50 cm de la estructura + 2 x 14 cm de las hélices)

En [10] se explica detalladamente todo el proceso de construcción, así como el presupuesto de un dron diseñado y construido por un aficionado. Este proyecto es notablemente más barato que el anterior, debido a que utiliza menos sensores. Esto, en teoría, se traduce en una calidad de vuelo menor, pero en el mismo enlace se habla de métodos y algoritmos de corrección para compensar la falta de precisión de algunos sensores. Concretamente, tiene un tamaño de 78 x 78 cm (50 cm de la estructura + 2 x 14 cm de las hélices) y un coste total de 101,50 €

2.3.10 *Resumen*

En la Tabla 2.2 se recogen las principales características de los proyectos citados anteriormente de los que se dispone de información:

Proyecto	Descripción	Dimensiones	Peso (+ carga)	Precio
Pelican	Potente y equipado con cámara	-	630 +650g	4000 $
MultiWii	Hecho a partir de componentes de mandos comerciales	Según diseño (78 x 78 cm aprox.)	Según diseño	120 €
Arducopter	Placa Arduino especializada y shield de sensores avanzado	Según diseño	Según diseño	400 €
Quad casero	Opción económica y de diseño muy abierto	78 cm	300 + 800g	102 €

Tabla 2.2.- Tabla comparativa de algunos modelos descritos anteriormente.

Como se ha comentado anteriormente, el modelo Pelican es muy usado en proyectos de investigación, ya que está equipado con sensores de altas prestaciones, lo cual se traduce en que sea muy estable y tenga un coste elevado.

En función de la aplicación o de calidad de vuelo deseada, los multicópteros pueden contar con más o menos sensores. Como mínimo son necesarios giróscopos de tres ejes, pero para un vuelo aceptable se utilizan también acelerómetro (también de tres ejes). Para aumentar la precisión e implementar ciertas funciones, se utilizan magnetómetros (para el control del ángulo yaw), barómetros (para el control de altura), sensores de proximidad, GPS, etc.

2.4 Componentes

Realizando el estado del arte se puede ver que, aunque puedan tener usos y configuraciones diferentes, todos tienen en común los siguientes elementos básicos:

- Estructura.
- Sensores (giróscopos y acelerómetros como mínimo).
- Motores y palas (y variadores, si los motores son trifásicos).
- Batería.
- Placa controladora.

Además, la gran mayoría cuenta con un receptor de radio para el mando de control remoto, pero hay otras formas de controlar el dron, como aplicaciones para Smartphone, a través de módulos Bluetooth, módulos WiFi, etc. También hay multicópteros que funcionan de forma autónoma. Por tanto, no es necesario el receptor de radio entre los elementos básicos.

A continuación se presentan los diferentes componentes entre los que se puede optar y que es necesario definir para diseñar y construir un dron.

2.4.1 Estructura

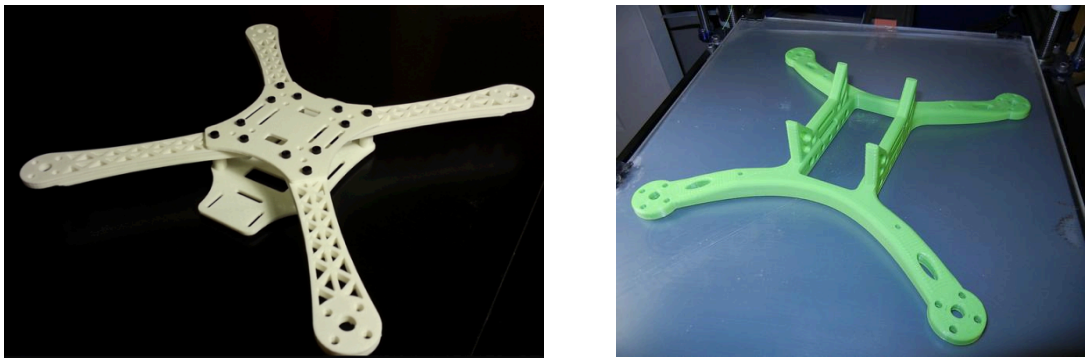
Las estructuras o el chasis pueden comprarse directamente, o pueden fabricarse fácilmente. Las comerciales suelen estar hechas de materiales ligeros y resistentes, para proporcionar robustez con poco peso. Se pueden encontrar muchos modelos en algunas páginas Web, tales como: HobbyKing, eBay, Amazon, etc.

(a) (b)

Figura 2-12.- Fotografías de algunos modelos comerciales de estructuras. (a) Fpv250. (b) Chasis Bumblebee.

En cuanto a la fabricación propia, es fácil unir tubos de fibra de carbono a piezas de sujeción para montar un chasis completo, pero lo es más aún imprimir las piezas con una impresora 3D y ensamblarlas (ver Figura 2-13). Concretamente, si se dispone de una de

estas impresoras, por ejemplo una RepRap, se puede buscar un modelo virtual en la página http://www.thingiverse.com/ e imprimirlo.



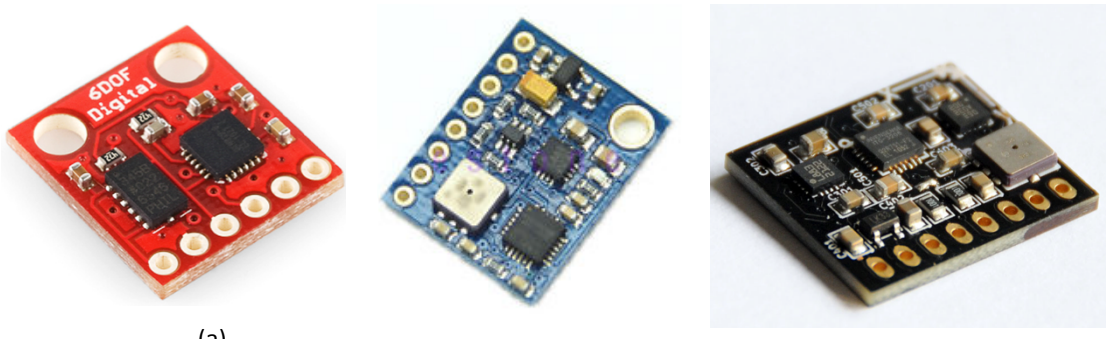
(a) (b)

Figura 2-13.- Estructuras obtenidas con impresoras 3D. (a) Mini Flame Wheel Frame. (b) 3D Printed quadcopter Frame.

2.4.2 Sensores

En cuanto a los sensores, es necesario combinar giróscopos y acelerómetros para poder estabilizar un multicóptero. En teoría, se puede hacer sólo con giróscopos, pero la precisión que se obtiene sin acelerómetros es bastante pobre, por lo que en este proyecto se utilizarán ambos.

Existen placas (IMU) que integran acelerómetros de tres ejes, con giróscopos, también de tres ejes; un eje para cada dirección en el espacio (Yaw, Pitch y Roll) (ver Figura 2-14).



(a) (b) (c)

Figura 2-14.- Placas IMU DOF. (a) 6DOF. (b) 10DOF. (c) Megapirate IMU.

También se pueden obtener estos sensores a partir de dispositivos que los integran, como es el caso de los controles de la consola Nintendo Wii (ver Figura 2-15). Estos controles llevan acelerómetros y giróscopos que pueden extraerse fácilmente, y conectarlos a la placa controladora del dron.

El accesorio Wii Motion Plus incorpora un giróscopo de tres ejes, Y el Wii Nunchuck lleva un acelerómetro de tres ejes. La combinación de estos dos elementos constituye una IMU como las descritas más arriba.

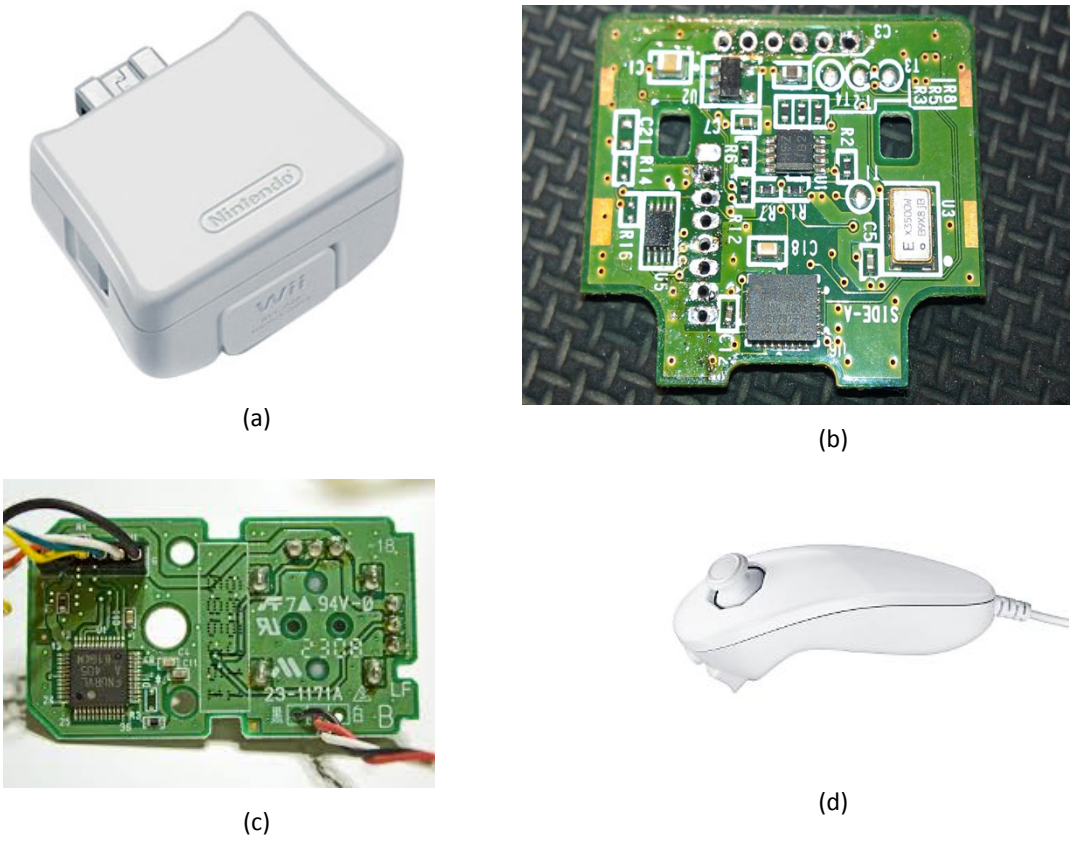


Figura 2-15.- Sensores de la videoconsola Wii. (a) Wii Motion Plus. (b) Placa WMP. (c) Placa WN. (d) Wii Nunchuck.

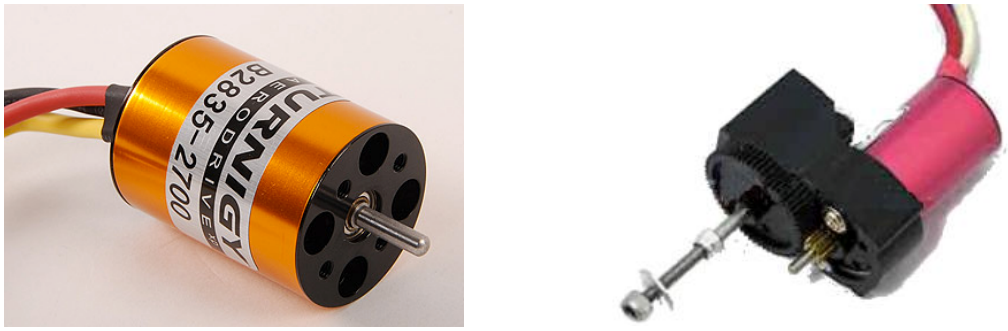
2.4.3 Motores y palas

Los motores y palas que se vayan a utilizar dependen del peso que se desee levantar, es decir, del peso del conjunto-dron. El empuje necesario se consigue con una correcta combinación motor-pala. Los fabricantes de motores especifican las palas adecuadas para conseguir potencias determinadas, lo cual simplifica la elección: se elige un motor de suficiente potencia, y se selecciona la pala que, según el fabricante, da el empuje necesario.

Dependiendo del peso del multicóptero, se utilizan motores de continua (menor potencia), o motores trifásicos con variadores de frecuencia, que transforman la corriente continua de la batería en una señal trifásica. Como el cuadricóptero que se pretende construir en este proyecto es relativamente grande, se utilizarán motores trifásicos sin escobillas, más conocidos como motores "brushless", los cuales generan más potencia que los motores de corriente continua.

Existen dos tipos de motores brushless: inrunner (rotor interno) y outrunner (rotor externo). Los inrunner proporcionan una mayor velocidad de giro, con un par bajo, por lo

que se utilizan con reducción mecánica. Los outrunner se usan sin reducción, ya que generan más par torsor, con una velocidad menor.



(a) (b)

Figura 2-16.- Motores brushless inrunner. (a) Sin reducción. (b) Con reducción mecánica.

Para evitar complicaciones mecánicas, se utilizarán motores outrunner. Se pueden encontrar fácilmente modelos de diferentes potencias. Algunas de las principales marcas son Turnigy, NTM y Scorpion.

(a) (b)

Figura 2-17.- Motores brushless outrunner. (a) NTM Prop Drive Series 28-26 1100kV. (b) Turnigy Multistar Elite 22-04 2300kV

En cuanto a las palas, lo único que hay que considerar es que, a mayor superficie, mayor empuje y menor velocidad de giro. Los motores brushless tienen un mayor rendimiento a revoluciones bajas, por lo que la mejor elección será unas palas del mayor radio posible.

Para aumentar aún la superficie de sustetación, pueden utilizarse hélices tripala en lugar de las bipala (ver Figura 2-18).

(a) (b)

Figura 2-18.- Hélices para aeromodelos. (a) Bipala. (b) Tripala.

2.4.4 Baterías

Las baterías son un elemento muy a tener en cuenta, no solo por ser la fuente de energía del dron, sino porque es el elemento más pesado del sistema (exceptuando las cámaras de alta calidad), y determina la autonomía. Al elegir la batería hay que tener en cuenta varios parámetros:

- Tensión: ha de ser acorde a la requerida por los motores.
- La capacidad: se mide en mAh (miliamperios por hora).
- La capacidad de descarga: mide la intensidad instantánea que es capaz de suministrar. Ha de ser mayor que la demanda máxima del sistema.
- Número de celdas: los motores brushless sólo admiten cierta cantidad de celdas de batería. Depende del modelo.

En la Figura 2-19 se muestra una batería de alta capacidad:

Figura 2-19.- Batería Gens ace 10000mAh 22.2V

En la etiqueta de la batería se da toda la información relevante:

- Tensión: 22.2V
- Capacidad: 10000mAh
- Capacidad de descarga: 25c (25 amperios máximo).
- Número de celdas: 6

Además de estos parámetros, vale la pena elegir una buena marca, que garantice una vida útil larga y una buena fiabilidad.

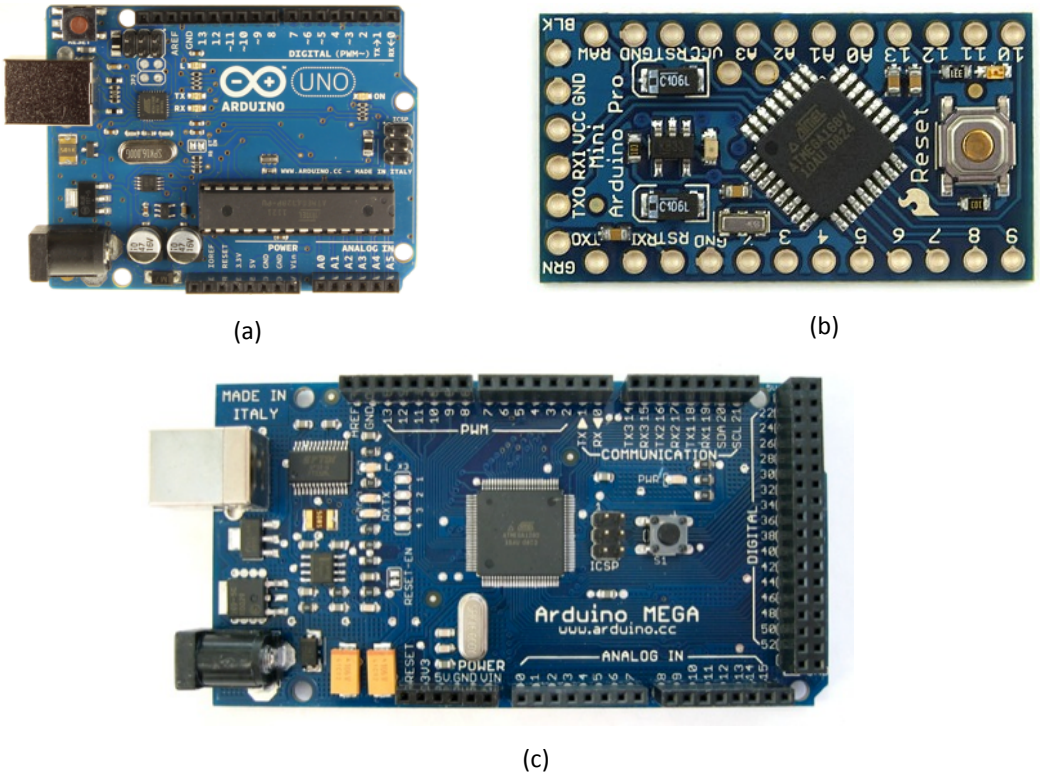
2.4.5 Placas controladoras

Pueden diferenciarse dos tipos de placas para drones, las básicas y las integradas. Las primeras incluyen sólo el microcontrolador o el microprocesador, por lo que es necesario añadir los sensores e implementar el código. Las integradas son placas comerciales que se venden con los sensores de movimiento integrados y con el código de vuelo ya cargado, por lo que tan sólo hay que conectarle los motores, acoplarla al chasis y ya pueden volar. A continuación se presentan los principales modelos.

2.4.5.1 Arduino

Arduino es una compañía que fabrica placas con el mismo nombre, para hacer fácil la construcción y programación de dispositivos de cualquier tipo. Están pensadas para acercar el mundo de la electrónica a cualquier persona, sin necesidad de que el usuario tenga grandes conocimientos de electrónica, por lo que sus características no son las más potentes del mercado. La compañía ha desarrollado diferentes modelos de placas Arduino y shields, los cuales se acoplan directamente a la placa principal, proporcionando funcionalidades extra, como comunicación por Bluetooth, WiFi o 3G, joysticks, buzzers, etc. Todas estas placas se programan con el entorno de programación Arduino IDE.

Algunos de los principales modelos se muestran en la Figura 2-20:

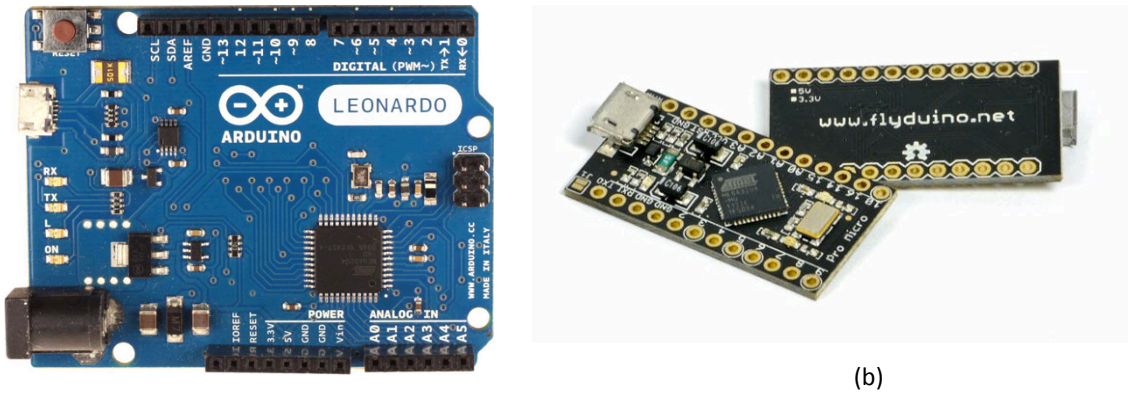


(a) (b)

(c)

Figura 2-20.- Placas Arduino. (a) Arduino UNO r3. (b) Arduino Pro Mini. (c) Arduino Mega.

Existen también placas fabricadas por otras empresas con los mismos microcontroladores, son los denominados "clones" de Arduino (ver Figura 2-21). Estas placas son compatibles con el entorno de programación Arduino IDE.



(a) (b)

Figura 2-21.- Placa Arduino Leonardo (a) y su clon Flyduino Pro Micro (b)

La principal diferencia entre las placas Arduino está en el microcontrolador que utilizan. Por ejemplo, Arduino Leonardo utiliza un chip ATmega32u4, Arduino UNO r3 utiliza un ATmega328 y Arduino Mega un ATmega1280. Estos chips aportan cada uno más pines digitales y analógicos que el anterior. La selección depende de la cantidad de dispositivos que se quiera conectar a la placa.

2.4.5.2 Raspberry Pi

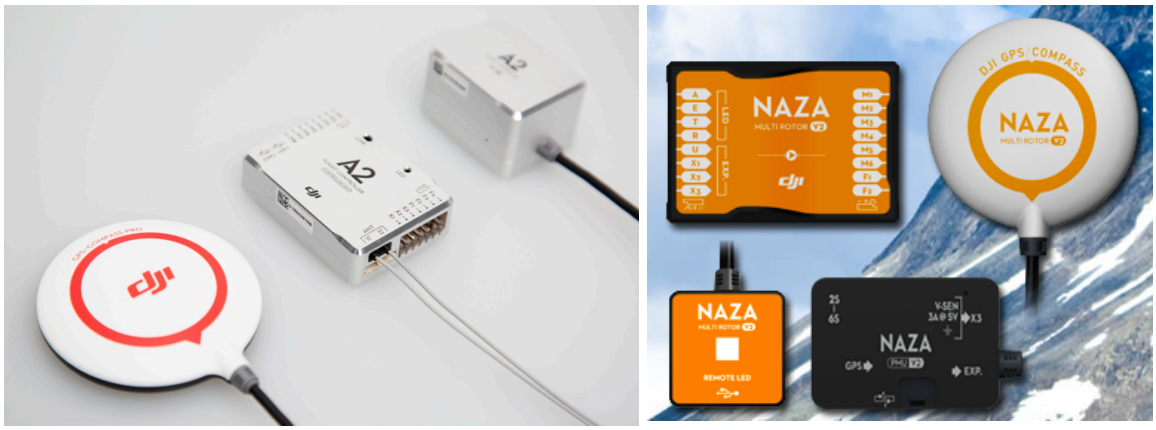
Las placas Raspberry son más potentes que las de Arduino, ya que llevan una un procesador en lugar de un microcontrolador. Son un ordenador en miniatura, con capacidad para realizar funciones como el procesado de texto, carga de juegos, o reproducción de video de alta calidad. Tiene integrados modos "Turbo", para realizar overclock de hasta 1GHz sin perder la garantía. Pueden utilizarse para proyectos de electrónica como este, ofreciendo una capacidad de trabajo superior.

En cuanto al software, puede llevar diferentes sistemas operativos: Raspbian, Arch Linux ARM y Pidora. Promueve el lenguaje de programación Python, pero también puede programarse en BASIC, C, Perl o Ruby.

Figura 2-22.- Raspberry Pi

2.4.5.3 *Placas DJI Innovations*

La marca DJI ha desarrollado placas controladoras, pensadas específicamente para multicópteros, que ofrecen un gran rendimiento y unos resultados muy precisos. Entran dentro de las placas integradas comentadas anteriormente, ya que se vende en un kit que incluye la IMU, el GPS y más complementos, además de traer cargado el software de vuelo, con varias funcionalidades extra, como la posibilidad de configurarse para 9 tipos diferentes de multicóptero, el control de orientación inteligente (IOC), el "punto de interés", (POI), para describir trayectorias circulares alrededor de un punto, la función "auto-vuelta a casa", y la protección frente al fallo de un motor. Ofrece comunicación por CAN BUS.



(a) (b)

Figura 2-23.- Kits de control DJI Innovations. (a) Kit DJI A2. (b) Kit DJI NAZA-M V2

2.4.5.4 *HobbyKing KK2.1.5*

HobbyKing ofrece una placa integrada, con IMU, pines para el receptor de radio y para el GPS, con software de vuelo preparado. Incluye un LCD para acceder fácilmente al menú de configuración. El chip que incluye en un microcontrolador ATmega644PA, similar a los utilizados por Arduino. Lleva implementadas funciones extra, como el control automático de altura.

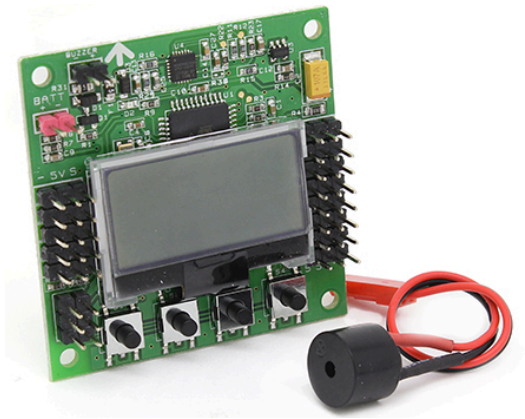


Figura 2-24.- HobbyKing KK2.1.5

Es una alternativa más económica a las placas DJI, que ofrece una fácil instalación y un buen funcionamiento.

2.4.5.5 Selección

Ante la variedad de placas que hay en el mercado, la elección depende de los objetivos del trabajo. Como se pretende implementar un software de vuelo propio, se utilizará una placa básica, para añadir posteriormente los sensores necesarios. Esto nos limita la elección a Arduino o a Raspberry. Al no ser necesaria una gran potencia de cálculo, se optará por una placa Arduino.

2.5 Conocimientos necesarios

Para integrar todos estos elementos, hay que seguir un proceso de diseño determinado: la aplicación determina la estructura y los sensores necesarios y, por tanto, el peso. El peso total determina la combinación de motores y palas necesaria. Finalmente, la suma de sensores y motores determina la placa que hay que utilizar, la cual debe tener los pines digitales y analógicos necesarios para todos ellos.

Este proceso de diseño requerirá los siguientes conocimientos:

- Conocimientos de electrónica, para saber qué sensores son necesarios para la aplicación del dron.
- Conocimientos básicos sobre sustentación de las palas.
- Definición del sistema de ecuaciones formado por la combinación de las fuerzas de sustentación de los cuatro motores.
- Conocimientos de programación, para crear los algoritmos de vuelo y la aplicación móvil de control remoto.

En los siguientes apartados se enumeran brevemente los componentes a utilizar.

2.6 Aplicación, sensores y estructura

La aplicación del cuadricóptero de este trabajo es la de vuelo teledirigido, por lo que no son necesarios sensores de visión, cámaras o sensores de posicionamiento avanzados, como GPS, barómetros, etc. Sólo se utilizarán sensores de movimiento (giróscopos y acelerómetros), siguiendo el proyecto Multiwii:

- Wii Motion Plus: 3 giróscopos.
- Wii Nunchuck: 3 acelerómetros.

Con estos sensores se puede medir con precisión los ángulos Pitch y Roll, pero el ángulo Yaw es más complicado de obtener, debido a que, al variar éste, no hay cambios en la detección de la aceleración de la gravedad en ningún eje del acelerómetro. Puede hacerse una estimación con el giróscopo para hacer pequeñas correcciones, pero será menos precisa que la de los otros ángulos.

La estructura será impresa con impresora 3D tipo RepRap, con una envergadura de 52 cm.

En cuanto a la comunicación, aunque lo más habitual sean los receptores de radio, incorporaremos un módulo bluetooth para conectar el dron con un dispositivo móvil. Aunque el bluetooth tenga un alcance limitado (unos 10m), nos vale para las pruebas de calibración y para vuelos cortos o en interiores.

2.7 Palas y motores

La combinación pala-motor es la que consigue una fuerza de sustentación determinada. La sustentación proporcionada por un perfil aerodinámico viene dada por la ecuación:

$$L = \frac{1}{2}\rho V^2 A C_L$$

Con:

L es la fuerza de sustentación en newton (= peso total / 4).

ρ es la densidad del fluido (aire) en kg/m^3.

V es la velocidad en m/s.

A es la superficie alar, en m^2.

C_L es el coeficiente de sustentación, adimensional, que depende de la hélice.

En esta ecuación la velocidad viene en m/s (lineal), por lo que habría que hacer una transformación a parámetros angulares. La principal utilidad de esta ecuación, en este caso, es la de mostrar de qué parámetros depende la sustentación. Como, según se ha dicho antes, los motores tienen un mejor rendimiento a revoluciones bajas, por lo que es mejor maximizar la superficie alar y el coeficiente C_L. Esto se consigue con hélices del mayor tamaño y paso posibles para la configuración de nuestro dron. Las hélices tripala proporcionan un mayor empuje que las bipala, por tener más superficie alar.

2.8 ESCs y tarjetas de control

Los ESCs (variadores) son los elementos electrónicos que se encargan de transformar el voltaje de la batería en señales válidas (trifásicas) para los motores trifásicos. Van conectados a la tarjeta de control para que ésta le indique la cantidad de potencia que tiene que mandarle a los motores.

La principal característica de los ESCs es la corriente máxima que pueden suministrar, que ha de ser mayor que la corriente para la máxima potencia en los motores.

En cuanto a la tarjeta de control, como se ha dicho antes, debe tener los pines suficientes para todos los elementos que se van a integrar. Tenemos:

- 4 motores (4 pines digitales PWM).
- Conjunto de sensores de movimiento (2 pines digitales).
- Pines de alimentación y masa (común para todos los elementos).

La demanda de pines es mínima, cualquier placa valdría para nuestro cuadricóptero, por lo que elegimos la de menor tamaño y peso posible: la placa Flyduino (es una adaptación de la placa Arduino Leonardo para aeromodelismo).

Capítulo 3

Materiales y Métodos

3.1 Introducción

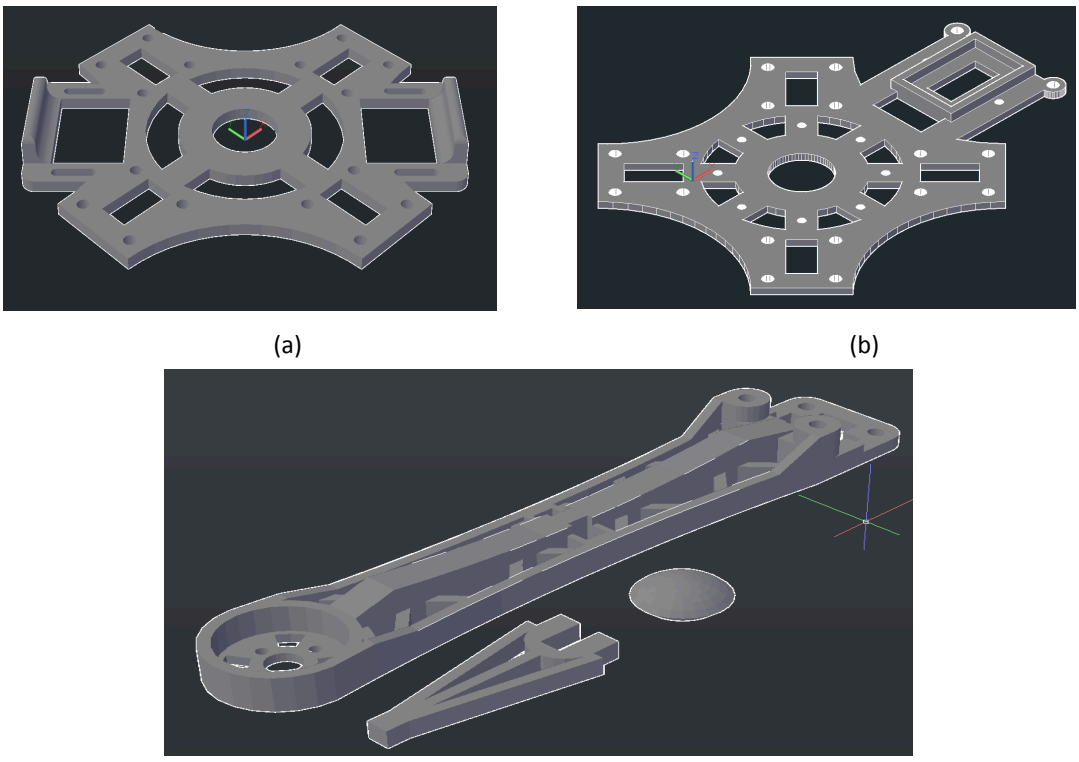
En este capítulo se describen detalladamente los elementos que componen el sistema, atendiendo a sus especificaciones técnicas, a los ajustes que se han realizado y al funcionamiento integral del conjunto.

Algunas características importantes de los elementos seleccionados se han comentado brevemente en el Capítulo 2. A continuación, se dan todos los detalles relevantes sobre cada uno, los cuales son:

- Estructura: Se ha llevado a cabo mediante impresión en 3D (52 cm de envergadura).
- Sensores: Se han utilizado los sistemas de interacción natural Wii Motion Plus y Wii Nunchuck.
- Módulo de comunicación: Se ha optado por usar el módulo Bluetooth Shield v2.2 de Itead Studio.
- Batería: Max Power de 11,1 V y 2100 mAh.
- Motores y palas: Brushless Outrunner.
- Placa controladora: Se ha seleccionado el módulo Flyduino Pro Micro.

3.2 Estructura

Sobre la estructura, cabe destacar que se ha seleccionado un modelo virtual inicial basado en los que se muestran en el portal web thingverse.com. A partir del modelo seleccionado, se han llevado a cabo un conjunto de pequeñas modificaciones con el fin de integrar adecuadamente el conjunto. Una de las ventajas de imprimir la estructura es que muchos de los modelos que se encuentran en la página son compatibles entre sí, por lo que se pueden coger partes de diferentes modelos y montar un chasis personalizado. Las piezas seleccionadas pueden verse en la Figura 3-1:

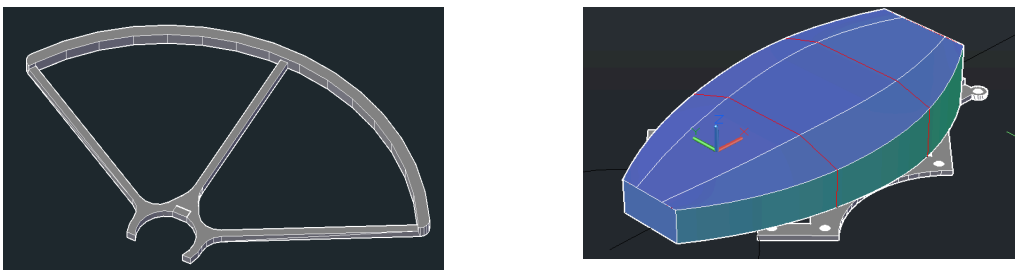


(a) (b)

(c)

Figura 3-1.- Estructura básica. (a) Cuerpo, parte inferior. (b) Cuerpo, parte superior. (c) Brazo y apoyo.

Para conseguir un aspecto final más elegante y proporcionar protección a la palas, se ha modificado el modelo de un protector de hélices, y se ha diseñado una cabina con Autocad. La Figura 3-2 muestra ambas piezas:



(a) (b)

Figura 3-2.- Partes adicionales. (a) Protector de las hélices. (b) Cabina.

A continuación se describe brevemente el funcionamiento de la impresora RepRap. La máquina calienta plástico ABS en un extrusor y lo deposita en una "cama caliente" para reducir el impacto térmico. La complejidad del proceso radica en el lugar en que deposita la materia prima (plástico, polímero u otra), y la forma en que lo hace. El proceso se controla mediante el uso de una aplicación software de código abierto, que toma como entrada el archivo de un modelo virtual, debidamente modificado, y genera las órdenes apropiadas que se envían a los motores.

En la Figura 3-3 se puede ver la impresora 3D RepRap, con el extrusor en la parte superior, movido por una polea unida a un motor paso-a-paso; y la "cama caliente", abajo, sobre la que se imprime. El movimiento horizontal se consigue combinando el movimiento del extrusor y de la cama. El movimiento vertical, se realiza con el tercer motor, el cual acciona los tornillos sinfín, a los lados del marco.

Las dimensiones máximas del objeto impreso están li mitadas por las de la cama. Ésta es de 200x200 mm, y el área de impresión queda en 180x180 mm. La altura está limitada por el marco, alcanzando los 180mm.

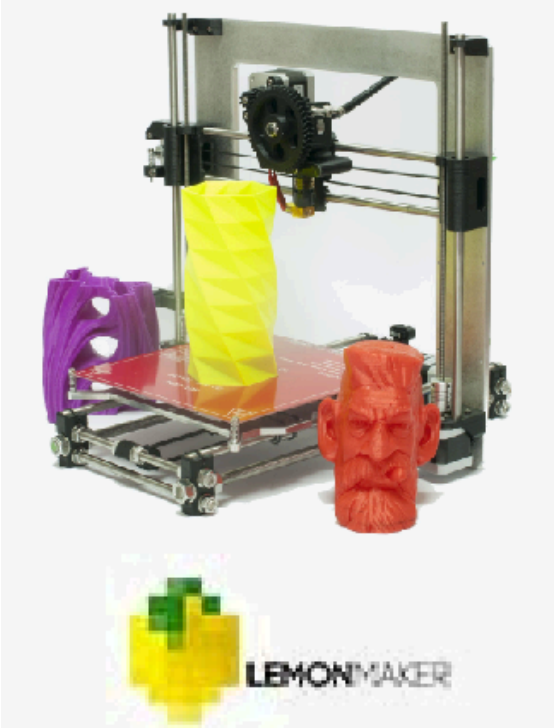


Figura 3-3.- Impresora 3D RepRap.

El procedimiento de impresión, en cuanto al procesado de los modelos y a la configuración de la impresora, se explica a continuación.

A partir del modelo virtual, se obtiene un archivo con extensión .stl. Los principales programas de CAD tienen esta opción en el menú de exportación de modelos, como es el caso de AutoCAD o Rhinoceros. En la exportación de los modelos del cuadricóptero, se ha visto que la de AutoCAD genera archivos mucho menos pesados (en memoria), pero el resultado es de menor calidad en las zonas redondeadas. Por el contrario, con la exportación de Rhino ocurre el proceso contrario. Sin embargo, en el resultado final no es importante una gran precisión, por lo que, para trabajar más rápido, se ha utilizado AutoCAD.

En Internet, es posible encontrar tanto archivos de extensión stl como modelos de CAD en formato editable.

Los archivos .stl han de ser transformados por medio de un programa de impresión 3D llamado Slic3r, a través del cual se fijan los parámetros de impresión, como la altura de las capas de ABS, el relleno de la pieza (en %), la temperatura del extrusor y de la cama

caliente, entre otros; y se le da a la pieza un formato en capas, para que la impresora lea el modelo en el orden en que lo imprime.

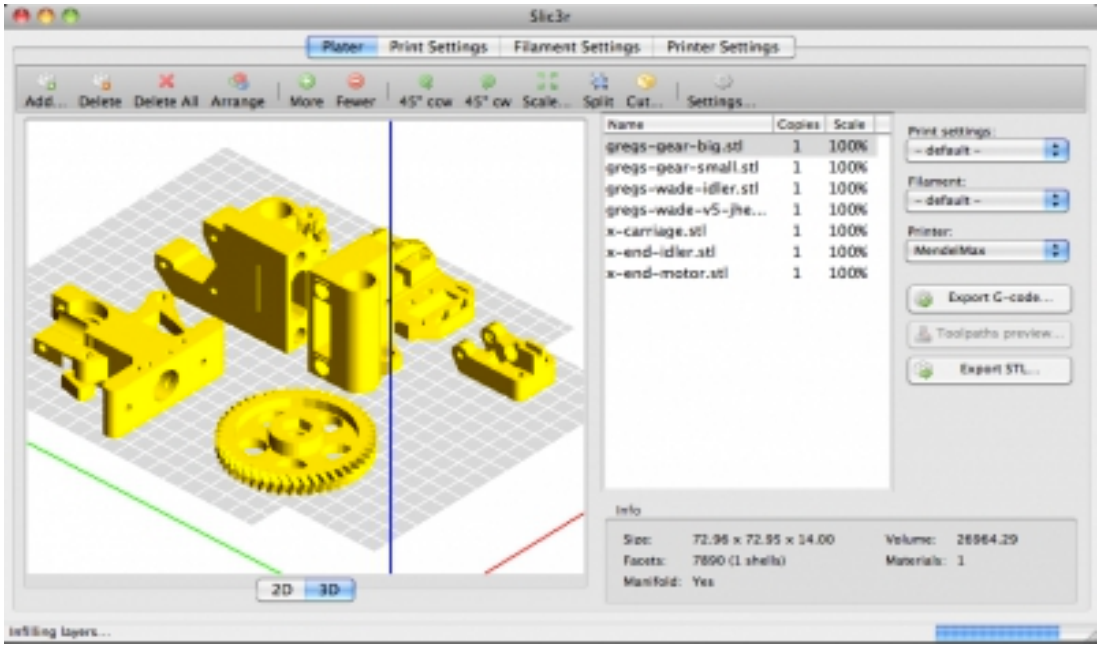


Figura 3-4.- Interfaz de Slic3r.

Slic3r genera un archivo en formato .gcode, listo para ser interpretado por la impresora.

El principal inconveniente de la impresión 3D es el tiempo de impresión. Para conseguir, por ejemplo, uno de los brazos del cuadricóptero, es necesario esperar unas cinco horas, por ser una pieza grande y con más relleno. Para las piezas de protección de las palas, la impresión dura casi dos horas. Dependiendo del tamaño, el tiempo de impresión varía bastante.

Finalmente, las piezas se fijan unas a otras y a los motores mediante tornillos.

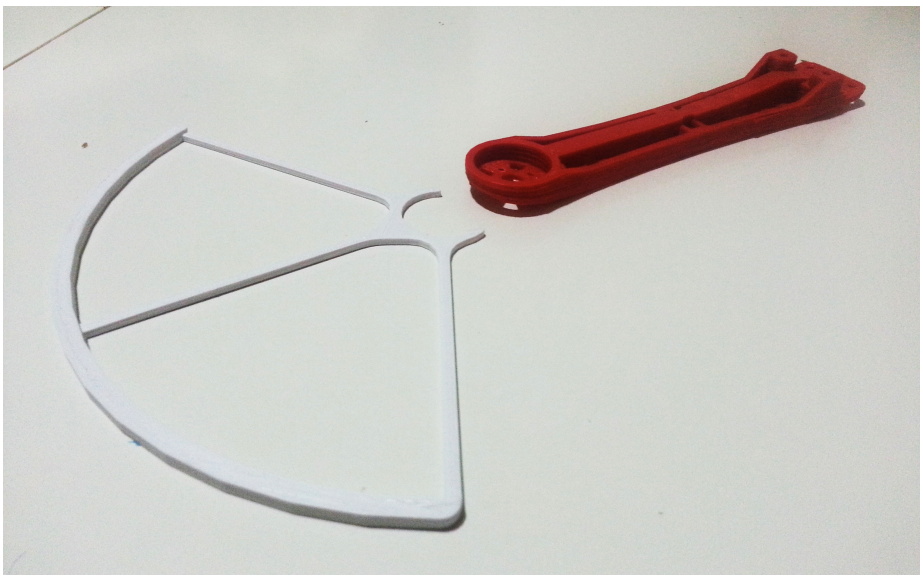


Figura 3-5.- Uno de los brazos con su protector.

El peso final del chasis queda en 250 g, repartidos entre la estructura, las piezas de protección de las palas y la cabina.

3.3 Sensores

Como se ha comentado anteriormente, los sensores a utilizar son los que se encuentran en el proyecto Multiwii, que consisten en una placa Wii Motion Plus, con giróscopo de tres ejes, y una placa de Wii Nunchuck con acelerómetro también de tres ejes.

3.3.1 Wii Motion Plus (WMP)

Wii Motion Plus es una extensión para el mando de la consola Wii de Nintendo. Consiste en un grupo de tres giróscopos adicionales que, junto con los tres acelerómetros del mando o del Nunchuck, permiten medir con precisión los movimientos realizados, tanto angulares como lineales.

La extensión Wii Motion Plus se conecta en la parte inferior del mando Wii Mote:

(a) (b)

Figura 3-6.- Accesorio Wii Motion Plus. (a) Conectado a un Wii Mote. (b) WMP solo.

Para su utilización en un cuadricóptero se elimina la carcasa (ver Figura 3-7).

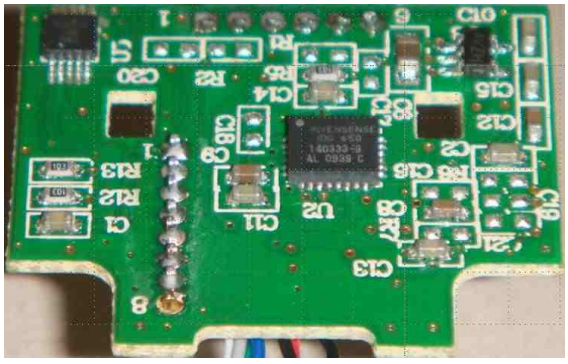
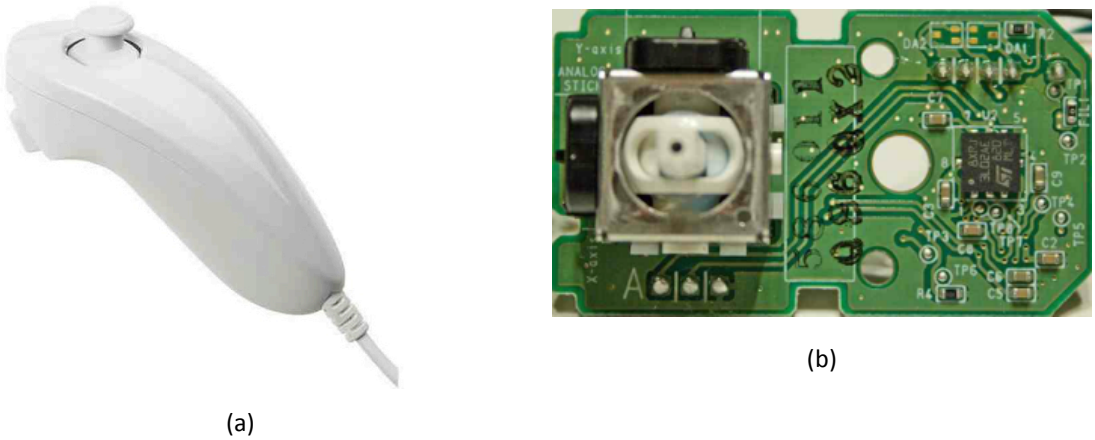


Figura 3-7.- Placa de WMP.

Esta placa incorpora giróscopos IDG650 para los ejes X e Y, y un ISZ650 para el eje Z, con una precisión de 2000 °/s. Los giróscopos dan señales analógicas, que se convierten en señales digitales útiles mediante adaptadores integrados en la propia placa. Este sensor transmite las señales digitales mediante el protocolo I^2C, soportado por las placas Arduino. La conexión es sencilla: se conectan los pines SDA (datos) y SCL (señal de reloj) de los sensores con los de Arduino.

3.3.2 Wii Nunchuck (WN)



(a) (b)

Figura 3-8.- Wii Nunchuck. (a) Accesorio con carcasa. (b) PCB.

El Wii Nunchuck es otra extensión para el mando de Wii, que ofrece un acelerómetro de tres ejes tipo LIS3L02AL. También se conecta a la parte inferior del mando de Wii y, cuando está conectado el Wii Motion Plus, se conecta a éste por su parte inferior. La placa, que puede verse en la Figura 3-8, incorpora un joystick y dos botones que no serán utilizados, por lo que se puede cortar la mitad de la placa (la de la izquierda en la imagen), dejando sólo los acelerómetros.

En la utilización normal de los sensores, el Nunchuck va conectado en serie al WMP, transmitiéndose a la vez la información de ambos por el mismo puerto I2C. La forma nativa de decodificar la información de los sensores tiene en cuenta si está conectado sólo el WMP, sólo en WN, o ambos, leyendo de diferente forma en cada caso. Esto complica la lectura, por lo que es necesario conectar los sensores de una forma alternativa, para poder leer los datos de cada extensión de manera independiente.

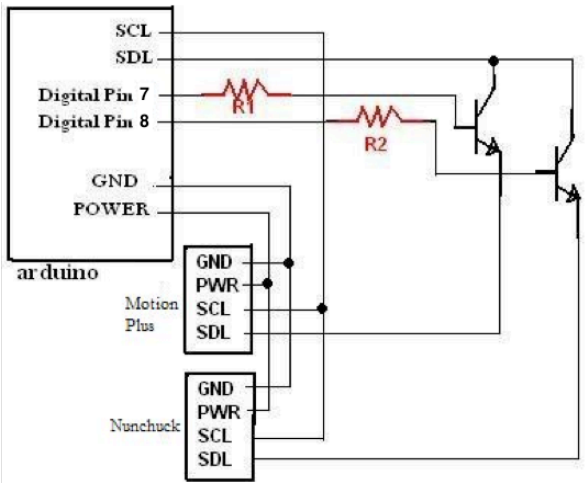


Figura 3-9.- Esquema de conexión.

El esquema de la Figura 3-9 permite sólo recibir datos del sensor al que se pida la información, por medio de dos transistores tipo 2N2222, y dos resistencias de 470 Ω. Controlando la señal de los pines de Arduino conectados a las bases de los transistores, se puede elegir de qué sensor se escucha la información en el pin SDA (en la imagen SDL).

Los sensores incrementan el peso del Cuadricóptero 15 g.

3.3.3 Decodificación de los sensores

Para leer los sensores de Wii, es recomendable implementar una librería con las funciones necesarias, simplificando el código de vuelo del cuadricóptero. Reuniendo la información necesaria, se ha creado dicha librería bajo el nombre de "WiiSensors". El código está disponible en un Anexo al final del documento.

Los sensores proporcionan lecturas de aceleración lineal y angular, a partir de las cuales se obtienen las medidas de los ángulos Yaw, Pitch y Roll. Primero, es necesario filtrar las lecturas en bruto, para evitar las interferencias por la vibración de los motores. Los multicópteros suelen usar un filtro de Kalman [11] para combinar la información proveniente de diversos sensores (giróscopo y acelerómetro), con la finalidad de obtener una estimación del ángulo lo más precisa posible desde un punto de vista probabilístico. Primero, se usa la función arco tangente de las aceleraciones angulares de dos ejes del acelerómetro, para obtener una primera medida del ángulo. Dicho valor se usa como entrada en el filtro de Kalman, junto con la que proporciona el giróscopo a partir de la medida de aceleración angular.

Al igual que la librería "WiiSensors", la librería "Kalman" se explica más detalladamente en el Anexo mencionado anteriormente.

3.4 Módulo de comunicación: Bluetooth shield

El módulo a utilizar es un shield Bluetooth de Itead Studio. Consta de pines de alimentación a 3,3 y 5 V. Tiene un interruptor con dos modos: CMD y DAT, para iniciarse en los modos "Command" y "Data", respectivamente. En el primero, el módulo puede recibir comandos para configurar distintos parámetros (nombre, visibilidad, velocidad en baudios, entre otros), y en el segundo envía y recibe información de manera normal por puerto serie. El módulo puede operar como maestro o como esclavo y tiene un peso de 20 g.

Figura 3-10.- BT Shield v2.2 de Itead Studio.

El funcionamiento es muy simple. Por defecto viene configurado con el nombre "itead", el pin "1234" y con una velocidad de 9600 baudios. No es necesario cambiar esta configuración. Una vez conectado con Arduino, simplemente envía y recibe por Bluetooth los caracteres que éste escribe en su puerto serie asociado, para que los reciba el dispositivo de control, es decir, hace de puente entre el dron y el control, sin realizar ninguna modificación de la información que pasa por él.

Para la fijación del módulo al chasis se imprime una pieza a modo de adaptador, con las dimensiones de la placa, y agujeros para atornillarla al cuerpo.

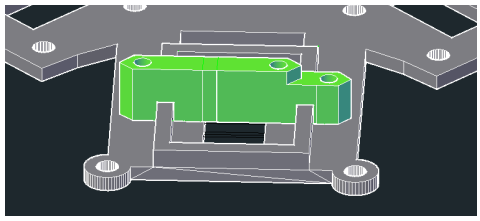


Figura 3-11.- Adaptador impreso para sujetar el módulo Bluetooth.

3.5 Batería

La batería escogida es de la marca Max Power, con una capacidad de 2100 mAh, tensión a 11,1 V, y un factor de descarga de 25 A. Es adecuado escoger una batería de calidad, ya que un suministro de potencia poco eficaz puede dañar todo el sistema tras un uso prolongado.

Figura 3-12.- Batería Max Power 11,1 V, 2100 mAh, 25C.

La marca Max Power ofrece buenas prestaciones, con una calidad y una fiabilidad excelentes.

3.6 Palas, motores y variadores

Los motores y las palas son el único elemento que no se ha comentado en el Capítulo 2, dado que requieren algo más de justificación teórica para su elección. Lo único pre-seleccionado es que se trata de motores brushless outrunner.

3.6.1 Selección y justificación

Es necesario elegir un conjunto pala-motor que proporcione la fuerza de sustentación suficiente para levantar el peso total del sistema. Además, el cuadricóptero ha de ser capaz de soportar una sobrecarga razonable, por lo que se mayora en un 100% el peso total, proporcionando un coeficiente de seguridad igual a 2.

Cuanto mayor sea e l peso del conjunto, más potentes y pesados serán los motores, por lo que se escogen inicialmente motores estándar para este caso: motores brushless outrunner de 150W, con un peso de 80 g (motor + variador). De esta manera, el peso total es de 780g, y el "peso simbólico", mayorado, queda en 1560 g.

Al haber cuatro motores, cada uno ha de levantar un cuarto del peso, es decir, 390g. Esto exige una fuerza de sustentación de 3,83 N por motor. Esta fuerza depende de la combinación pala-motor, por lo que existe más de una solución.

Viendo las características de diferentes motores, dadas por los fabricantes, se escogen los motores A2212 de 150 W, que proporcionan un empuje máximo de 6,37N cada uno, con hélices de 9x6. Esto supone un coeficiente de seguridad de:

Figura 3-13.- Motor A2212, 1000 kV, 13 T.

$$n = \frac{6,37}{3,83} = 1,66$$

Este coeficiente de seguridad, está calculado sobre el peso mayorado al doble del real, por lo que se puede asegurar que, con las palas de 9x6, los motores tienen fuerza para levantar 1,66 veces el doble del peso del dron.

Por tanto, se utilizan los motores brushless outrunner A2212 1000 kV, 13T, de 150 W.

Los motores deben ir conectados a variadores de frecuencia, los cuales deben ser capaces de suministrar la intensidad suficiente. Para una potencia de 150 W, con un rendimiento del 80%, a 11,1V, se demanda una intensidad de 16,89 A. Teniendo en cuenta los picos de intensidad en el arranque, son adecuados unos variadores de 30 A, de RC Hobbies (Figura 3-14).

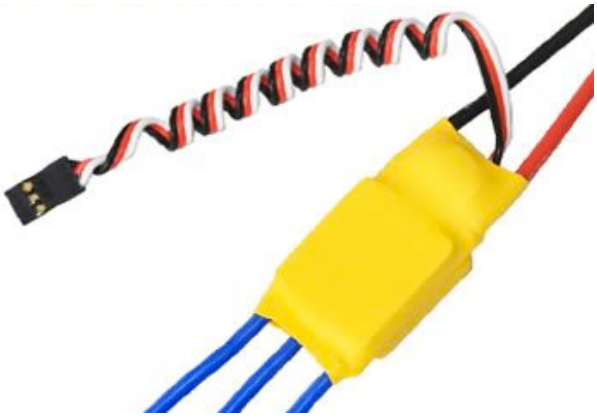


Figura 3-14.- RC Hobbies 30 A ESC

3.6.2 Funcionamiento de los motores

En primer lugar, es necesario explicar el funcionamiento de los variadores. Éstos tienen un pequeño firmware cargado para accionar el motor, y permiten configurar algunos parámetros mediante un menú implementado con pitidos, el cual está pensado para trabajar con un receptor de radio, leyendo la palanca del empuje (throttle) hacia arriba o hacia abajo. Para acceder al menú, se enciende el variador (conectado al motor) con la palanca del mando arriba. Los parámetros configurables son los siguientes:

- Freno: Activado / Desactivado. Por defecto: Desactivado.
- Tipo de batería: Li-xx / Ni-xx. Por defecto: Li-xx.
- Protección frente baja tensión: Cortar / Reducir. Por defecto: Reducir.
- Umbral de baja tensión: Bajo / Medio / Alto. Por defecto: Medio.
- Modo de arranque: Normal / Suave / Muy suave. Por defecto: Normal.
- Temporización: Baja / Media / Alta. Por defecto: Media.

Sólo es necesario cambiar el modo de arranque, a "Muy suave", el cual es el idóneo para multicópteros.

En este trabajo no se utiliza un receptor de radio, pero las posiciones de la palanca del mando se traducen en valores PWM, enviados por el pin blanco de los variadores. En el entorno de programación de Arduino, esto se realiza mediante la función `analogWrite()`, la cual escribe en un puerto PWM un valor que depende del ancho de pulso de una señal cuadrada. Este valor puede tomar valores en el rango de 0 a 255. La sintaxis es: `analogWrite(pinMotor, valor)`.

Desde el menú de configuración de los variadores, también es posible configurar el rango de la palanca del mando de control, para que puedan funcionar con distintos mandos. Por tanto, es posible configurar el rango de valores que se mandan con `analogWrite()`. Se configuran unos límites de 80 a 254, ya que los valores por debajo de 80 son considerados como "fuera de rango" por los variadores, emitiendo éstos los correspondientes pitidos.

El procedimiento para arrancar los motores es el siguiente:

- Se manda un 80 con `analogWrite()` para inicializar el motor (el variador lo interpreta como joystick hacia abajo).
- El motor responde con un pitido tipo 123 indicando que la tensión de alimentación es la correcta.
- Luego, emite tres pitidos , uno por cada celda de batería conectada; y un pitido largo de confirmación.
- Ahora, el motor está listo para arrancar. Basta con enviar valores mayores para ir consiguiendo más potencia (el umbral a partir del cual los motores arrancan está en `analogWrite(pinMotor, 100)`).

Antes de conectar todos los motores, es conveniente realizar ensayos para saber qué intensidad máxima pueden demandar, comprobándose que la batería escogida es capaz de suministrar tal intensidad sin dañarse. Esto validará o descartará la batería escogida.

3.6.2.1 Ensayo en vacío

En primer lugar, se realiza un ensayo en vacío, es decir, sin ninguna pala acoplada al motor. De esta manera se comprueba que el motor se mueve correctamente y que las conexiones están bien realizadas, además de obtenerse datos para comparar con los del siguiente ensayo, con pala.

La conexión es simple: los tres cables del motor se conectan a los tres del variador, y éste se conecta a la batería y los pines de Arduino de alimentación, masa y PWM. En la Figura 3-15 se muestra el esquema de conexión con una placa Arduino UNO r3.

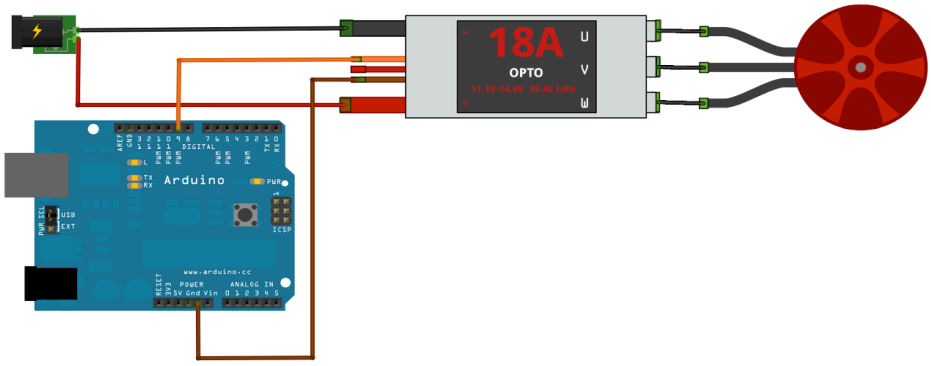


Figura 3-15.- Conexión motor-variador-Arduino.

Los valores de intensidad en un motor en el ensayo sin pala se dan en la Tabla 3-1:

analogWrite()	I (A)
91	0,33
100	0,47
110	0,50
120	0,58
130	0,65
140	0,69
150	0,71
160	0,73
170	0,74
180	0,74
190	0,74
200	0,74
210	0,74
220	0,74
230	0,74
240	0,74
250	0,74
254	0,74

Tabla 3-1.- Ensayo sin pala.

Como puede observarse, el motor admite valores de hasta 254 con el comando `analogWrite()`.

Para valores inmediatamente superiores a 254, el motor se para automáticamente, del mismo modo que si se le mandase un 80, ya que la función `analogWrite()` espera valores de tipo byte entre 0 y 255, y por tanto, al enviar un valor mayor de 255, que necesita más de 8 bits, se trunca a los 8 bits menos significativos. Para no desbordar la función, se truncan los valores introducidos mediante la función `constrain()`, la cual corta los valores que se salen de los límites superior e inferior que se especifiquen.

3.6.2.2 Ensayo con pala

Este ensayo es igual que el anterior, con las mismas conexiones, pero con una pala acoplada al motor. Al montarla, ésta ofrece una mayor resistencia, generándose más intensidad en el motor. La potencia depende de cada valor PWM que se le mande.

Los valores de intensidad obtenidos, para cada valor de la señal de control, se recogen en la Tabla 3-2:

analogWrite()	I (A)
91	0,40
100	0,48
110	1,12
120	1.87
130	2,47
140	3,20
150	4,26
160	4,20
170	4,28
180	4,27
190	4,20
200	4,23
210	4,26
220	4,20
230	4,20
240	4,20
250	4,20
254	4,20
255	Paro

Tabla 3-2.- Ensayo con pala.

Igual que en el ensayo en vacío, la intensidad llega a su máximo a partir de 170 en el analogWrite(), pero alcanzando los 4,28 A. Este aumento de intensidad se debe, como se ha comentado antes, a la resistencia que ofrece la pala al mover el aire.

Hay que tener en cuenta que estos valores son estacionarios. Durante el arranque se registran intensidades de 6 A.

Se realiza un tercer ensayo, también con pala, pero situando una superficie plana justo debajo del motor, para simular el efecto que el suelo puede producir en el movimiento del aire arrastrado por la pala y, por tanto, en la fuerza que el motor necesita para moverlo. Se obtienen unos valores similares a los del ensayo anterior, por lo se puede afirmar que el suelo no produce ningún efecto significativo en la intensidad demandada por los motores.

Por tanto, se puede concluir este apartado al establecer que el valor de intensidad máxima, estacionaria, es de 4,28 A por motor; y el de pico es de 6 A. Entre los cuatro motores la intensidad demandada máxima es de 24 A. La batería seleccionada es de tipo 25C, por lo que no habrá ningún problema de exceso de intensidad. Las baterías 25C aguantan corriente estacionarias de 25 A sin problema, y de hasta 50 A de pico con riesgo de dañarse. La intensidad de 24 A de los cuatro motores es ya la de pico, lo cual deja bastante margen de seguridad.

3.6.3 Pruebas de empuje

En el apartado 3.5.1 se ha dado como dato el empuje de cada motor, según el fabricante, de 6,37N, para palas de 9x6. Para comprobar este valor, y para obtener datos con diferentes hélices, se realiza un cuarto ensayo. Leyendo el peso del conjunto con un robot de cocina, mientras los motores operan a diferentes velocidades, se puede medir el empuje de los motores.

Este ensayo se ha realizado con el cuadricóptero ya ensamblado. El montaje del mismo se describe detalladamente en el Capítulo 4. El procedimiento del ensayo es el siguiente:

El robot de cocina es capaz de medir hasta 2200 g, y sólo hacia abajo, como es lógico. El problema es que el empuje de los motores va hacia arriba y, por lo tanto, si se quiere medir, es necesario apoyar el cuadricóptero con un peso extra que proporcione el margen suficiente para que el peso total no se haga negativo. Se mide la diferencia de peso para cada velocidad de giro. De esta manera, se obtiene el empuje con una sencilla correlación:

$$L = -g \cdot (p_i - p_0)$$

Donde g es la aceleración de la gravedad, p_i es el peso de cada medida, y p_0 el peso inicial, con los motores apagados.

El cuadricóptero se ata al robot de cocina para que quede bien sujeto y se puedan accionar los motores con seguridad. El montaje puede verse en la Figura 3-16. Se han utilizado hélices de 8x4,5, un poco más pequeñas que las supuestas anteriormente. Esto reducirá el empuje supuesto en el apartado 3.6.1.

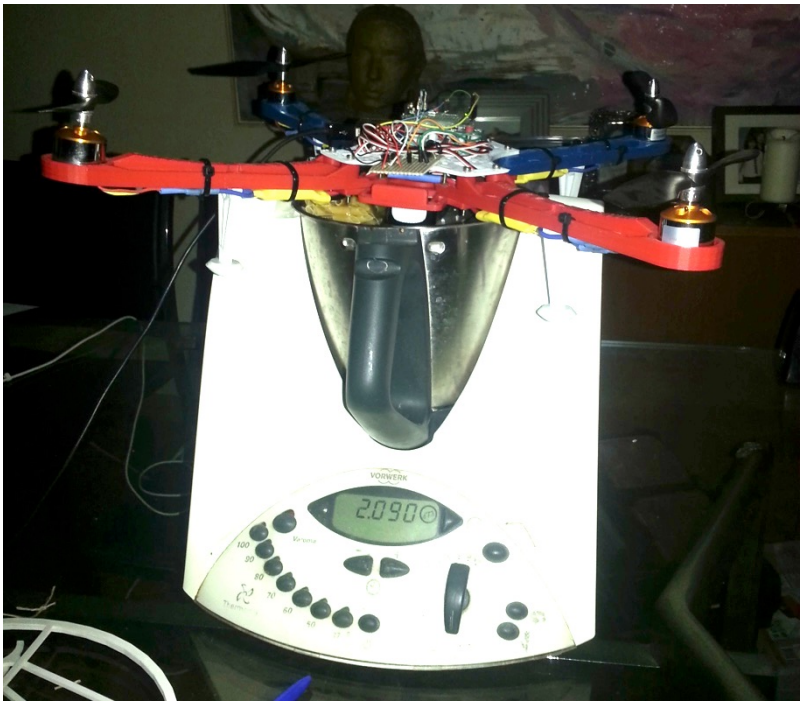


Figura 3-16.- Ensayo de empuje.

Para un peso inicial de 2090 g (peso del sistema más el lastre), se obtienen los siguientes valores:

analogWrite()	$p_i - p_0$ (g)	L (N)
80	0	0
100	-160	1,57
110	-250	2,45
120	-440	4,32
140	-610	5,98
160	-750	7,36
180	-900	8,83
190	-1030	10,10
200	-1220	11,97
210	-1420	13,93
220	-1670	16,38
230	-1890	18,54
240	-2080	20,40
254	-2260	22,17
255	0	0

Tabla 3-3.- Ensayo de empuje.

En la Figura 3-17 se muestran los resultados gráficamente.

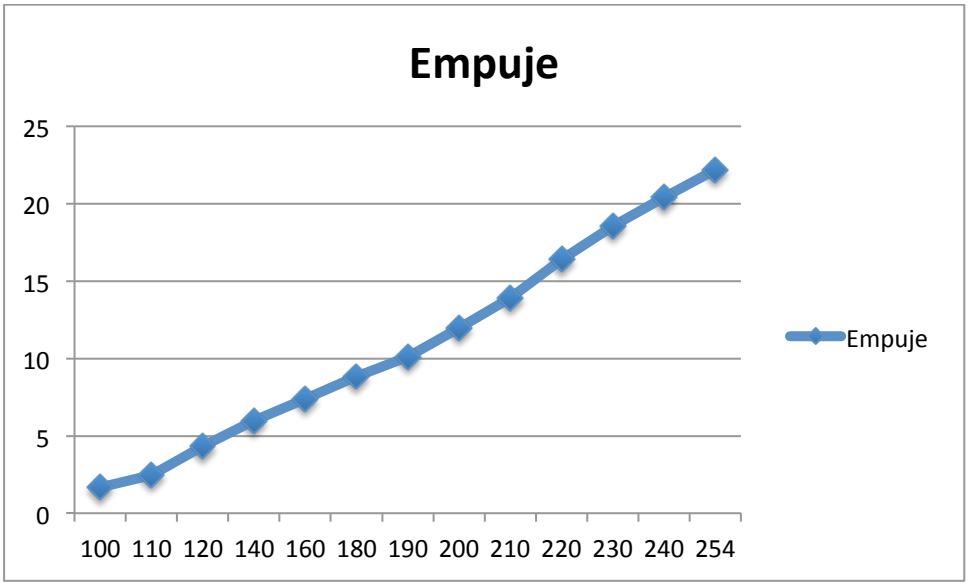


Figura 3-17.- Gráfica del empuje frente al valor de write().

Los resultados se aproximan a una recta, cuando el empuje está relacionado con el cuadrado de la velocidad de giro. Esto ocurre porque los variadores tratan la señal de control para enviarle potencia a los motores de forma controlada. Así, la variación del empuje al cambiar la señal de control es la misma para todos los motores.

El empuje máximo obtenido es de 22,17 N. Cada motor genera un cuarto de este empuje, es decir, 5,54 N. Es algo menor que los 6,37 N indicados en la sección 3.6.1, debido a las hélices utilizadas. El coeficiente de seguridad queda en:

$$n = \frac{5,54}{3,83} = 1,45$$

Por tanto, se comprueba que la combinación motor-pala elegida es completamente válida, permitiendo al cuadricóptero despegar con los motores a media potencia, pudiendo éstos levantar el doble del peso total del sistema, a máxima potencia.

Si se utilizan palas mayores, el empuje también será mayor, permitiendo al cuadricóptero despegar a menos revoluciones y levantar más peso.

Comentarios finales del ensayo:

- Se ha visto que el aparato despega para un valor de `analogWrite()` de 167 (media potencia, aproximadamente), quedando sujeto por los hilos en tensión.
- Al tomar el último valor, se supera el peso total inicial. Para tomar esta medida no es necesario interrumpir el ensayo, simplemente se añade más lastre (380 g más) y se resta después a la medida junto con el peso inicial.

3.7 Física del cuadricóptero

En cuanto a su funcionamiento como conjunto, un cuadricóptero se maneja normalmente de forma manual, pero la estabilización es totalmente automática. Sería bastante complicado para el piloto hacer correcciones con la suficiente velocidad como para conseguir un vuelo estable, por lo que en el software se implementa un regulador PID que realiza estas correcciones sin que el usuario tenga que preocuparse de ello. Con los sensores de movimiento se calculan los ángulos Yaw, Pitch y Roll, y se comparan con los ángulos que el piloto desea. La diferencia entre ambos, es decir, el error, se introduce en el PID, el cual consigue una señal óptima para los motores si está bien ajustado

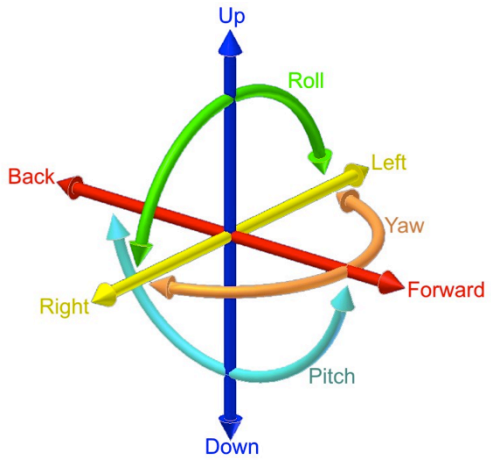


Figura 3-18.- Seis grados de libertad en un dron.

Los ángulos Yaw, Pitch y Roll se miden según se indica en la Figura 3-17. Contando con los movimientos lineales en los tres ejes, el sistema tiene seis grados de libertad.

Para moverse a adelante y hacia atrás, se modifica el ángulo Pitch, y para moverse hacia los lados se cambia el Roll. La magnitud en la que se modifica un ángulo es directamente proporcional a la velocidad de avance en la dirección correspondiente.

Hay que tener en cuenta que, siempre que se utiliza un motor, éste genera un par torsor que hace girar la hélice, pero también genera un movimiento del sistema en el sentido contrario. Es lo que hace que los helicópteros convencionales necesiten el rotor secundario, para equilibrar esta fuerza. Una de las ventajas del cuadricóptero, es que tiene un número de rotores par, con lo que el par torsor de un motor se compensa con el de otro. Por tanto, para que las fuerzas se compensen entre sí, dos motores deben girar en el sentido de las agujas del reloj, y los otros dos en el contrario, tal y como se muestra en la Figura 3-18.

Los sentidos de giro deben ir alternados, para tener control sobre el ángulo Yaw.

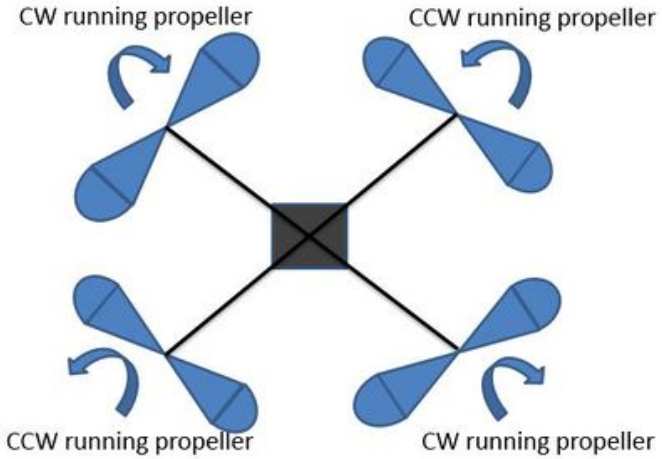


Figura 3-19.- Sentidos de giro de los motores.

Para moverse en dirección vertical, basta con incrementar o disminuir la velocidad de los cuatro motores. Para modificar uno de los ángulos, hay que incrementar la velocidad de la pareja de motores adecuada, disminuyendo la de la pareja opuesta. En la Figura 3-20 puede se ilustra qué parejas de motores varían cada ángulo.

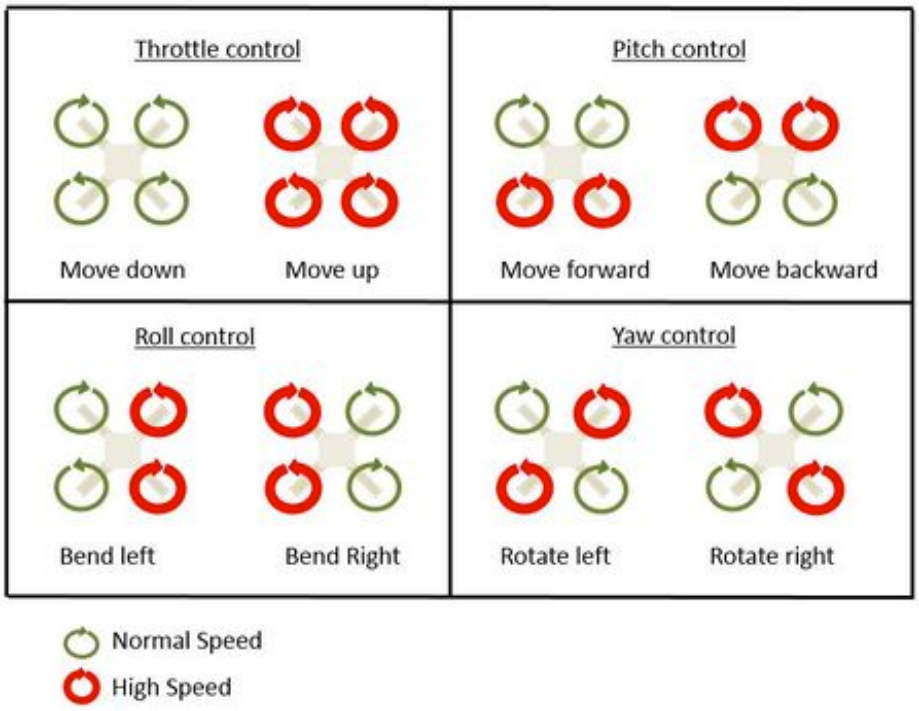


Figura 3-20.- Control del empuje y los ángulos Yaw, Pitch y Roll mediante parejas de motores.

Los motores señalados en rojo son los que aumentan la potencia, y los señalados en verde la disminuyen.

De esta manera, para tener un control total sobre los movimientos del cuadricóptero, se fija el valor del empuje (throttle), y se suman y restan los valores de Yaw, Pitch y Roll de salida que proporciona el PID. El fragmento de código que se encarga de mover los motores, según este criterio, es simple:

```
throttleFL = throttle - OutPitch + OutRoll - OutYaw;
throttleFR = throttle - OutPitch - OutRoll + OutYaw;
throttleBL = throttle + OutPitch + OutRoll + OutYaw;
throttleBR = throttle + OutPitch - OutRoll - OutYaw;
analogWrite(motorFL, throttleFL);
analogWrite(motorFR, throttleFR);
analogWrite(motorBL, throttleBL);
analogWrite(motorBR, throttleBR);
```

La denominación de los motores es:

- Front Left
- Front Right
- Back Left
- Back Right

Así definidos, el sentido positivo del ángulo Pitch es el que hace al dron avanzar hacia delante, y el sentido positivo del Roll, es el de movimiento hacia su derecha.

Capítulo 4

Diseño y montaje del Hardware

4.1 Introducción

En este capítulo se describen en detalle el Cuadricóptero que se ha diseñado, así como las fases que se han llevado a cabo para realizar el montaje completo del Cuadricóptero.

4.2 Recopilación de material

El material utilizado en el montaje del cuadricóptero, incluyendo materiales adicionales de soporte y sujeción son:

- Motores brushless outrunner A2212 1000kV 13t de 150W.
- Variadores de 30A.
- Chasis impreso en 3D de 52cm de envergadura.
- Sensores de movimiento de Wii: WMP y Nunchuck.
- Placa controladora Flyduino Pro Micro.
- Bluetooth Shield v2.2 de Itead Studio.
- 10 tornillos de acero inoxidable de M3.
- Herramientas: alicates, destornilladores, llave Allen, cúter, cables, termo-retráctil, bridas y conectores macho-hembra.

El procedimiento sigue el siguiente orden:

4.3.- Montaje de los motores en los brazos del chasis.
4.4.- Conexión de los variadores.
4.5.- Fijación de los cuatro brazos al cuerpo.
4.6.- Cableado de potencia.
4.7.- Conexiones variadores-Arduino.
4.8.- Montaje de los sensores y el módulo Bluetooth.
4.9.- Montaje de las hélices.
4.10.- Montaje de los elementos adicionales.

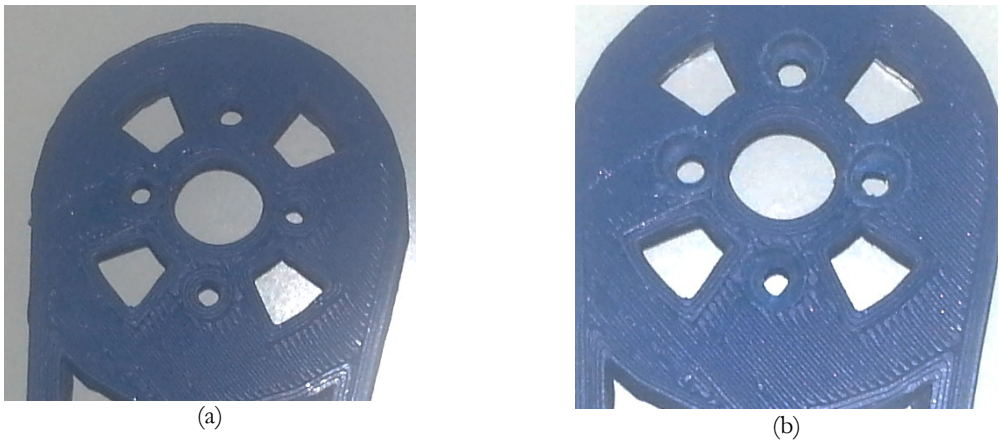
4.3 Montaje de los motores

En la siguiente figura se muestran todos los materiales y herramientas necesarias para realizar este montaje.

Figura 4-1.- Materiales para el montaje de uno de los motores.

Lo primero, es fijar los motores a los brazos del chasis. Para su sujeción, los motores vienen con unos tornillos cortos de cabeza cónica. Estos tornillos son demasiado cortos y no traspasan el grosor de la base de apoyo del motor, pero son difíciles de encontrar, ya que tienen una métrica poco común. La solución adoptada es la de taladrar unos milímetros el plástico del brazo con una broca de 5 mm, así el tornillo pasa y, además, se obtienen agujeros con la misma forma cónica de los tornillos.

En la Figura 4-2, se muestra el taladro realizado en los agujeros de uno de los brazos del drone. En la imagen de la izquierda se ve la pieza sin modificar, y en la de la derecha puede apreciarse el ensanchamiento de cada agujero.



(a) (b)

Figura 4-2.- Sujeción de los motores. (a) Antes del taladro. (b) Después del taladro.

Una vez hechos los agujeros, se atornillan los motores y se acoplan a las puntas de los cables, los conectores macho. Se repite el proceso para cada uno de los cuatro brazos. En la Figura 4-3 se puede ver uno de los brazos con el motor atornillado.

Figura 4-3.- Fijación del motor al brazo.

4.4 Conexión de los variadores

El siguiente paso es conectar los variadores a cada motor. Para ello, se fijan los conectores hembra a los cables del variador, se unen con los conectores de los cables de los motores y se protegen con termo-retráctil. Además, para fijar los variadores a los brazos del chasis se utilizan bridas.

Aquí es donde hay que tener en cuenta el sentido de giro de cada motor, ya que éste depende de la conexión de las fases. Para cambiar el sentido de giro, basta con intercambiar dos de las fases de un motor. De este modo, si al conectar un motor sin cruzar ningún cable, éste gira en el sentido de la agujas del reloj, al realizar el proceso anterior girará con el sentido opuesto.

En la Figura 4-4 se muestra la conexión del motor BR (Back-Right), el cual gira en sentido horario.

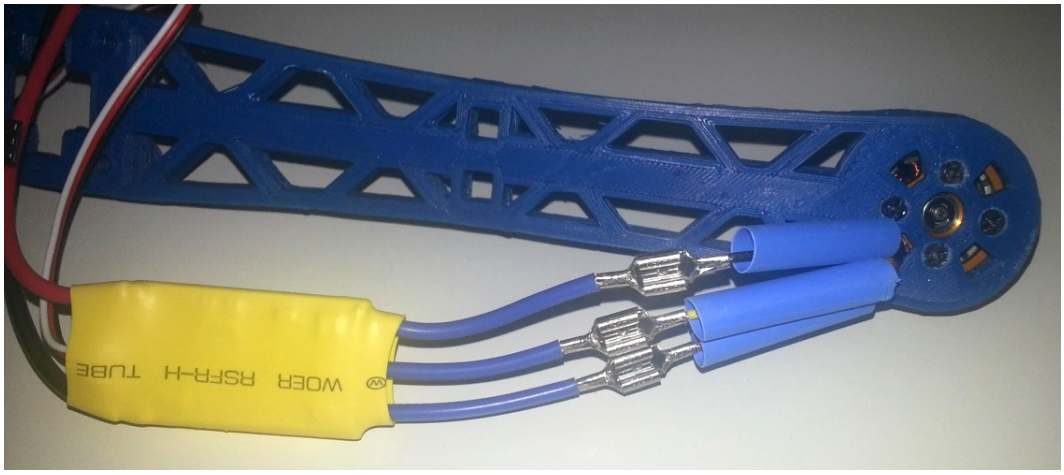


Figura 4-4.- Conexión motor-variador.

Los variadores conectados se fijan a los brazos mediante bridas, una para la placa y tres para los cables. Con este paso terminado, los cuatro brazos están ensamblados y listos para unirlos al cuerpo.

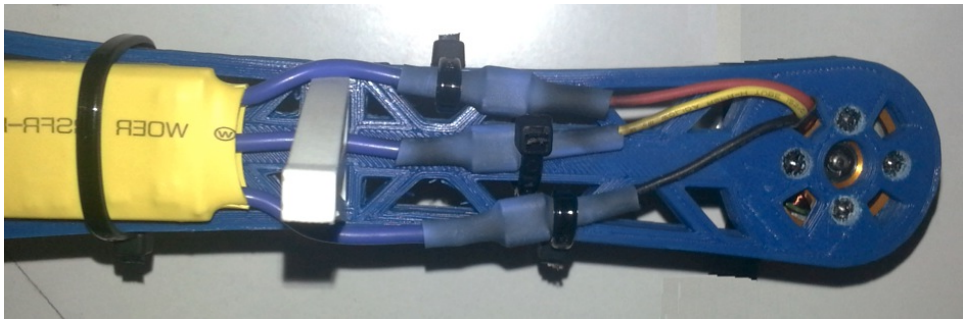


Figura 4-5.- Fijación de un variador al brazo mediante bridas.

4.5 Fijación de los brazos al cuerpo

Los brazos tienen agujeros para los tornillos de métrica 3, que permiten atornillarlos al cuerpo. Primero, se fijan a la pieza del cuerpo inferior para permitir acceder al interior del chasis, donde irá el cableado de potencia. Después, se fijará la pieza superior, dando rigidez a la estructura. En la Figura 4-6 se pueden ver los brazos atornillados al cuerpo inferior, con la pieza superior posicionada.

Figura 4-6.- Fijación de los brazos al cuerpo inferior.

4.6 Cableado de potencia

Una vez conectados a los motores, los variadores también han de conectarse a la batería y con Arduino. Para esto, deben estar presentados los cuatro brazos en la pieza del cuerpo inferior. Cuando el cableado de potencia esté listo, se añadirá la pieza del cuerpo superior dejando los brazos bien sujetos y posicionados.

Hay cuatro variadores y, por lo tanto, cuatro cables para un polo de la batería y cuatro más para el otro. Para unir los cuatro cables en uno solo, se pueden utilizar conectores en Y como el de laFigura 4-7, o se pueden utilizar conectores macho-hembra como los de los consiguiendo el mismo resultado. El esquema de conexión se muestra en laFigura 4-8.

Figura 4-7.- Conector en Y.

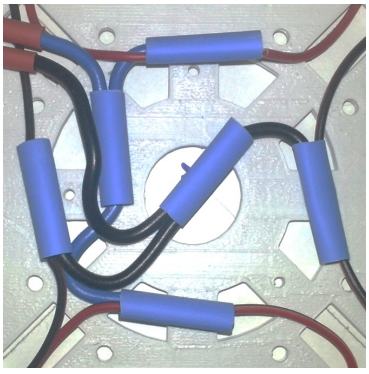


Figura 4-8.- Esquema de conexión de los variadores a la batería.

En cada unión hay dos machos introducidos el mismo conector hembra. Así, además de conseguir la conexión deseada, se queda muy bien unido y la conexión es segura. Se unen los positivos de dos variadores (dos cables rojos en uno azul), en un lado del drone. En el otro lado, se unen los rojos de los otros dos variadores. Luego, se unen los dos azules en uno solo, quedando conectados los cuatro positivos en un solo cable azul. Se hace igual para los negativos (cables negros). Cada unión va protegida con tubo termo-retráctil.

En los dos cables del final, se suelda un conector macho para la batería, que proporciona un contacto seguro y fiable (ver Figura 4-9).

Figura 4-9.- Conector macho para la batería

Es muy importante que no se invierta la polaridad en ningún variador. Las conexiones deben estar bien realizadas para que no se junten en ningún punto polos opuestos de variadores o batería. Los colores de los cables ayudan a no confundirse al realizar las conexiones, y el termo-retráctil protege frente a contactos no deseados.

4.7 Conexión variadores-Arduino

Con el cableado de potencia a punto, el único cable de los variadores que falta por conectar es que va conectado con Arduino (ver **Figura 4-10**). El cable negro va a masa, el rojo a VCC, y el blanco va a un pin PWM, para recibir la señal de control. En la placa Flyduino Pro Micro los pines correspondientes no están juntos, por lo que es necesario separar los cables del conector triple, reorganizarlos, e introducirlos en conectores vacíos, que sí coincidan con la disposición de los pines de la placa de control.

Figura 4-10.- Esquema de conexión variadores-Arduino.

El esquema de conexiones puede verse en la Figura 4-11. En gris, están los cables de las señales de control, en negro las masas y, en rojo, los cables de alimentación. Estos últimos, se utilizan para alimentar la placa de control desde los variadores. En realidad basta con conectar uno pero, si fallase el variador conectado, la placa entera se apagaría. Por esto, se conectan todos los cables en el mismo pin.

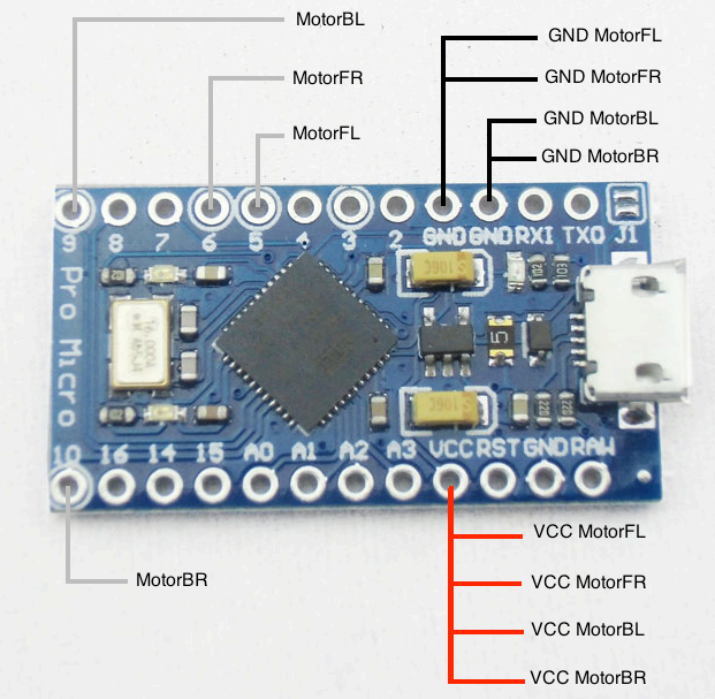


Figura 4-11.- Esquema de conexión variadores-Arduino.

Para que las conexiones queden sólidas, se utilizan los mismos conectores triples de los variadores, pero con los cables redistribuidos (ver Figura 4-12).

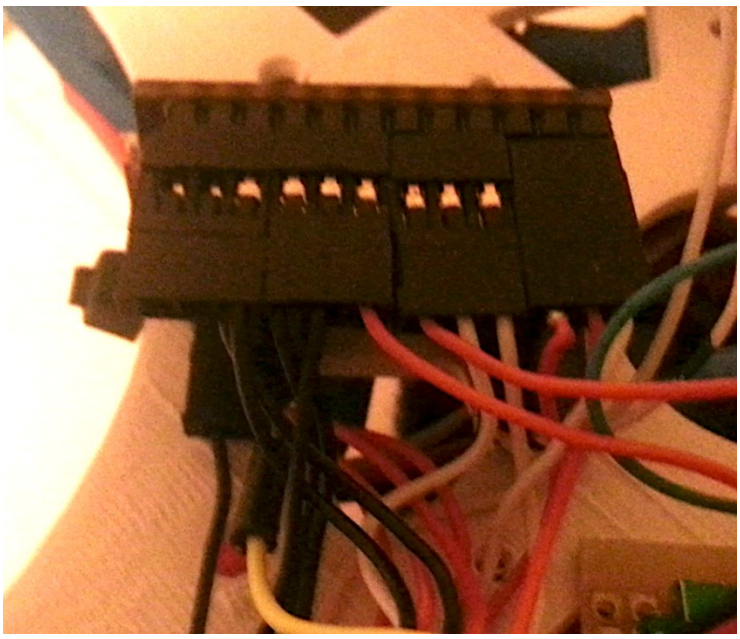


Figura 4-12.- Conexión a Flyduino con conectores triples.

4.8 Montaje de los sensores

El esquema de los sensores ya fue descrito en el Capítulo 3. Son necesarios dos transistores 2N2222 y dos resistencias de 470 Ω.

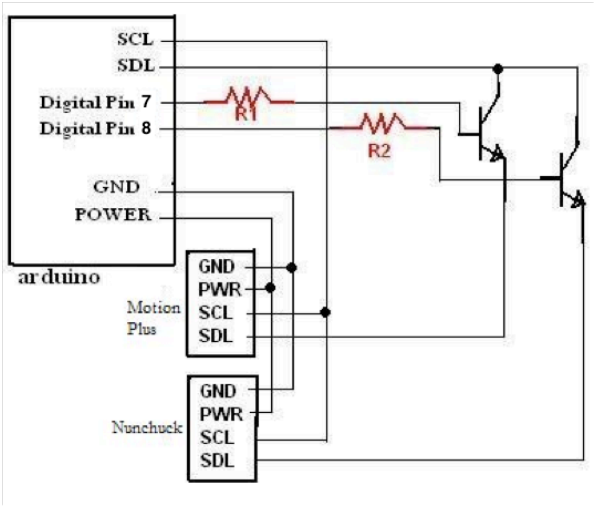
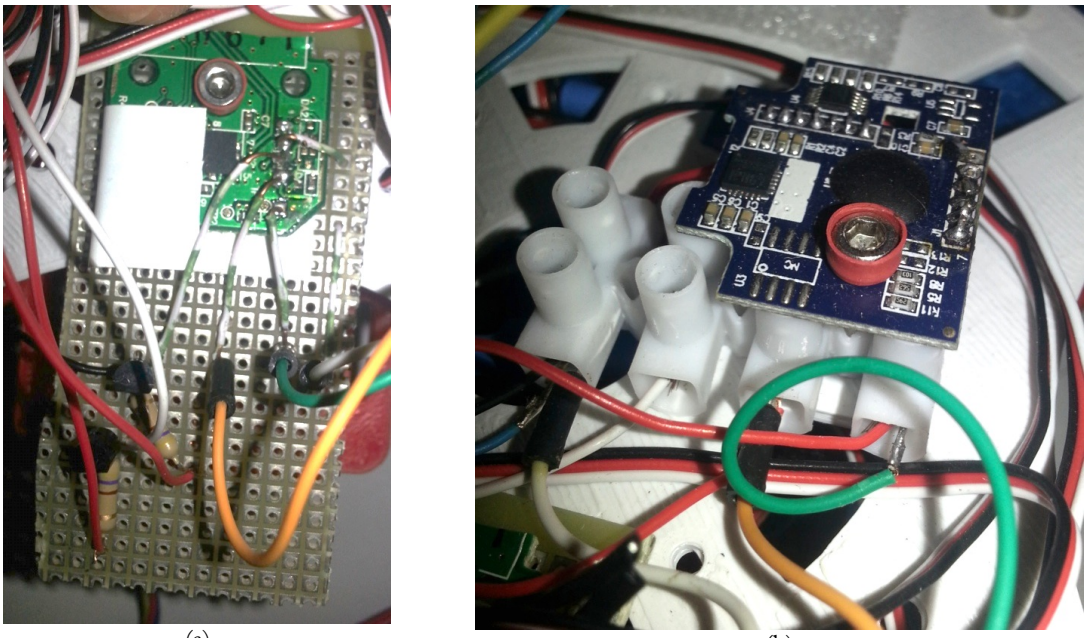


Figura 4-13.- Esquema de conexión de los sensores.

Los sensores deben quedar fijos a la estructura, ya que deben estar alineados con los ejes del cuadricóptero, y vibrar lo menos posible. Las placas de Wii se atornillan al cuerpo, junto con elementos de conexión necesarios. Para colocar los transistores y las resistencias se utiliza una placa perforada, donde también se coloca el Nunchuck. Para las conexiones de la placa WMP, se acopla una regleta (ver Figura 4-14).



(a) (b)

Figura 4-14.- Conexiones de los sensores de movimiento. (a) Placa del Nunchuck. (b) Placa WMP.

La conexión del módulo Bluetooth es bastante sencilla. Sólo hay que conectar cuatro cables: alimentación, masa, Tx (transmisión de datos) y Rx (recepción de datos). Para los cables de alimentación y masa se puede utilizar la regleta de la placa WMP, evitando así utilizar más conectores. Los pines Tx y Rx se conectan a Rx y Tx de Flyduino, respectivamente, con conectores hembra como los de los variadores. El módulo se fija al cuerpo mediante tornillos, con el adaptador descrito en el capítulo anterior.

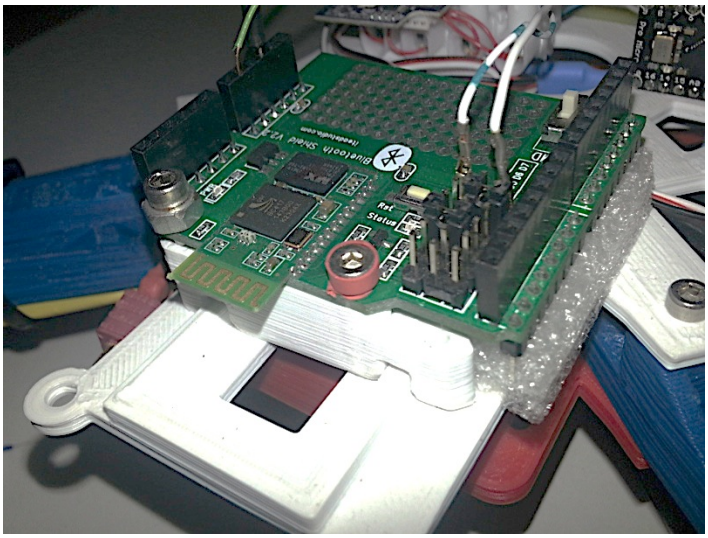


Figura 4-15.- Módulo Bluetooth acoplado al cuerpo.

Se ha colocado bajo el módulo un material blando, a modo de cama, que minimiza los esfuerzos mecánicos sobre la placa. Llegado este punto ya se tendría montada toda la electrónica.

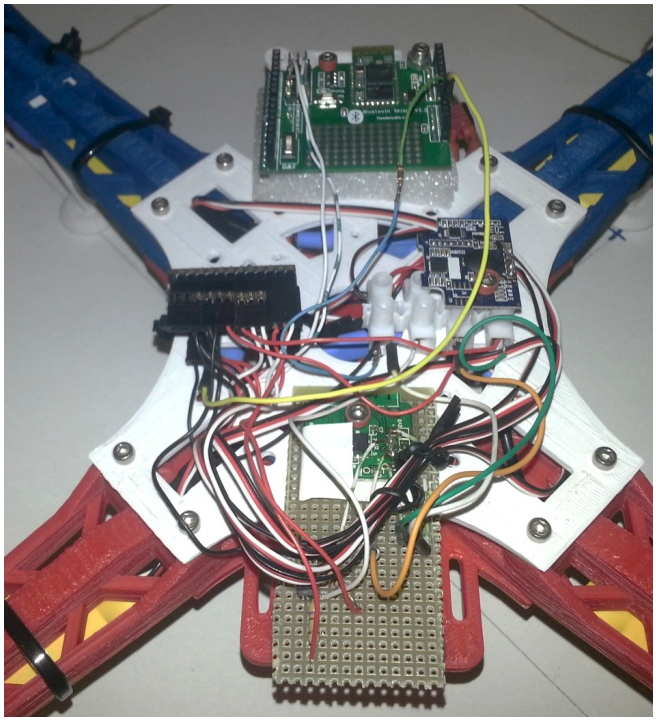


Figura 4-16.- Fotografía del montaje de la parte electrónica completa.

4.9 Montaje de las hélices

Los motores vienen con adaptadores para acoplar las hélices (ver Figura 4-17).Tan solo hay que desenroscar la parte superior, colocar la hélice, y apretar bien con un pasador introducido en la parte superior del adaptador.

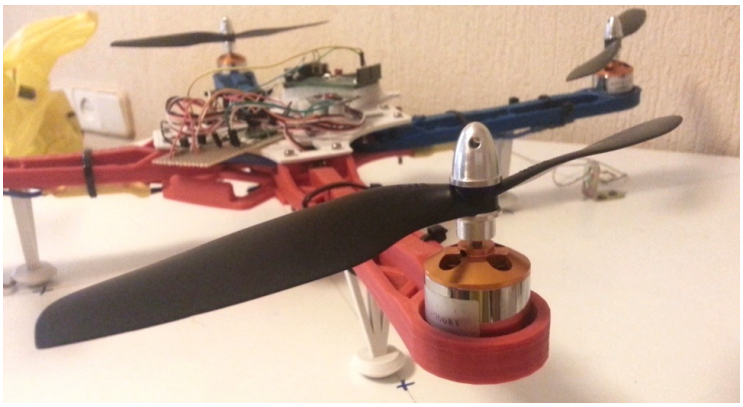


Figura 4-17.- Hélice acoplada al motor.

4.10Montaje de los elementos adicionales

Los elementos adicionales (patas, protectores y cabina) no son necesarios para el funcionamiento del drone, pero ofrecen protección frente a posibles impactos o aterrizajes forzosos.

4.10.1 Patas

Las patas están diseñadas para introducirse en los agujeros de los brazos. Se introducen por presión, y quedan lo bastante sujetas como para no tener que utilizar ningún adhesivo.

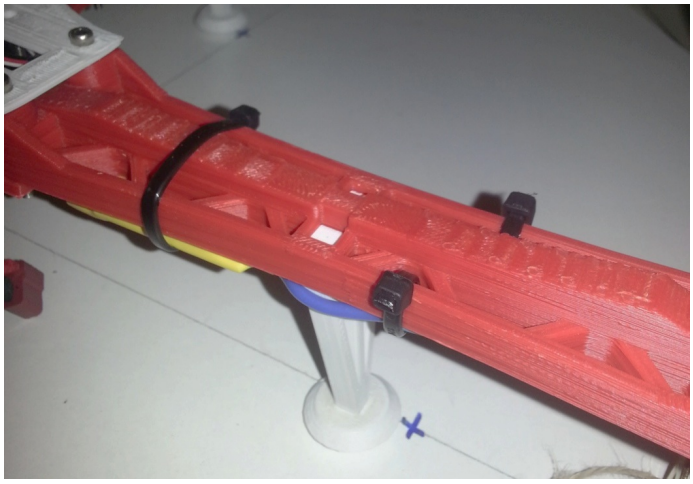


Figura 4-18.- Pata acoplada por presión a un brazo.

Al colocar todas las patas, el drone cojea un poco, por lo que se liman las bases de dos de las patas (ver Figura 4-19).

(a) (b)

Figura 4-19.- Bases de las patas. (a) Antes de limar. (b) Después de limar.

4.10.2 Protectores de las hélices

Los protectores de las hélices se pegan a los brazos con adhesivo. Han de acoplarse con cuidado para que ni el adhesivo ni el protector toquen el motor, ya la parte externa de éste es móvil, y rozaría.

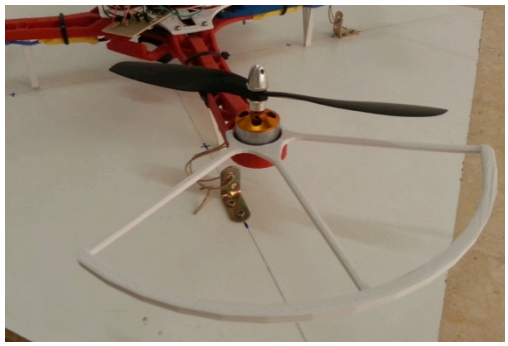


Figura 4-20.- Protector de una de las hélices

4.10.3 Cabina

La cabina es de diseño propio, para que encaje con la pieza del cuerpo superior, y no haya problemas de espacio con la electrónica. Se le ha dado altura suficiente para que no toque la parte más alta (el módulo Bluetooth), y pueda conectarse el cable USB a Flyduino por medio de un agujero lateral.

Figura 4-21.- Cabina

4.10.4 Sujeción para la batería

La batería se coloca debajo del cuerpo inferior ya que, al ser el elemento más pesado del sistema, desplazará el centro de gravedad del Cuadricópterohacia abajo, lo que se traduce en más estabilidad. Para colgarla, se pasa una cinta de velcro-adhesiva entre las ranuras de la pieza inferior (ver). La parte adhesiva fija la cinta al cuerpo, y el velcro abraza la batería, así se puede retirar y recolocar la batería fácilmente cuando sea necesario cargarla.

(a)

(b)

Figura 4-22.- Sujeción de la batería. (a) Velcro. (b) Batería sujeta.

Capítulo 5

Diseño del software de vuelo y calibración

5.1 Introducción

El control de un aeromodelo suele ser algo complicado. Los modelos convencionales de aviones o helicópteros no tienen sensores de movimiento que les ayuden a auto-estabilizarse, y todo el control depende del piloto. A diferencia de estos modelos, los multi-cópteros disponen de sensores que les permiten conseguir por sí mismos la estabilidad. Esto es sin duda una mejora respecto a sus antecesores, pero genera complicaciones de software que deben ser abordadas para conseguir dicha auto-estabilidad.

Dando una vista general del software de vuelo, se pueden diferenciar las siguientes partes:

- Lectura de los sensores: Recogida de datos sobre la posición del drone.
- Filtrado de los datos: Reducción de ruido existente en los sensores, para la obtención de unos datos más estables y fiables.
- Comparación de los datos de posición del drone con los valores deseados: La posición del drone debe ser la deseada por el piloto. La diferencia entre ambas es el error.
- Introducción del error en un regulador PID: El regulador produce la señal de control usada por los motores.
- Movimiento de los motores según las indicaciones del regulador.
- Control del usuario: El software debe leer e interpretar correctamente las órdenes del piloto de manera rápida y eficiente.

El movimiento de los motores y la interacción con el usuario no tiene grandes complicaciones. El grueso del código es el filtrado de los datos obtenidos de los sensores y el regulador PID. Para entender ambos elementos, es necesaria una introducción teórica.

5.2 Conceptos teóricos

La lectura de los sensores es específica para cada modelo. Los fabricantes proporcionan librerías para la lectura de sus productos. En el caso de los sensores de la videoconsola Wii, son los usuarios los que descifran las tramas propietarias enviadas por los sensores e implementan funciones y librerías para su utilización.

Los sensores a utilizar son un acelerómetro y un giróscopo, ambos de tres ejes. Los acelerómetros miden aceleraciones lineales y los giróscopos aceleraciones angulares. Para el desarrollo del código del drone sólo se necesitan lecturas de los ángulos Yaw, Pitch y Roll, por lo que podría parecer que no es necesario un acelerómetro. No obstante, las lecturas de los giróscopos acumulan errores enormes en pocos minutos. Por otro lado, lo acelerómetros, aunque se ven muy afectados por el ruido (de los motores, del viento, etc.), no acumulan error. Por tanto, al combinar los datos de ambos sensores, se pueden obtener valores válidos para los ángulos. A partir de los acelerómetros se pueden calcular los ángulos Pitch y Roll, por la variación del efecto de la gravedad sobre dos de los ejes del sensor. Es por tanto necesario, un algoritmo que compute las medidas de ambos sensores y filtre el ruido del acelerómetro.

5.2.1 Filtro de los sensores

Existen diferentes tipos de filtros para los sensores de movimiento. Los más utilizados son el filtro por matriz de cosenos directores, el filtro de Kalman y el filtro complementario. La matriz de cosenos directores es difícil de implementar, y el filtro complementario, aunque válido, no es tan preciso como el filtro de Kalman. Por tanto, se implementa el filtro de Kalman [11].

El filtro de Kalman es un algoritmo desarrollado por Rudolf E. Kalman, que sirve para estimar el estado no medible de un sistema dinámico lineal a partir de mediciones de ruido blanco[1]. Es un método recursivo que minimiza el error cuadrático.

Tiene dos etapas, que se repiten en cada iteración: predicción y corrección. En la etapa de predicción, se estima el estado futuro del sistema y de la matriz de covarianza. En la etapa de corrección se calcula la ganancia de Kalman, se corrige el estado del sistema con esta ganancia y la medida actual, y se actualiza la matriz de covarianza.

En la práctica, se puede decir que el filtro de Kalman realiza un filtrado, a partir de una estimación inicial (introducida por el usuario), y le aplica una corrección en cada

[1]Ruido blanco: Señal aleatoria con una densidad espectral de potencia "plana". se caracteriza porque sus valores de señal en dos instantes de tiempo diferentes no guardan correlación estadística.

iteración, proporcional al error entre el sistema actualizado y la estimación del estado futuro. En la Figura 5-1 se muestra en diagrama de bloques el algoritmo de Kalman.

En la aplicación para un drone, el estado del sistema que se mide es el ángulo. La predicción se realiza a partir de la medida del giróscopo, de la que se obtiene la velocidad angular (derivada del ángulo y, por tanto, el factor para obtener el valor futuro). El valor de la medición actual, que corrige la estimación en la segunda etapa, es la medida del ángulo del acelerómetro, obtenido a partir del arco-tangente, como se ha explicado en el apartado anterior.

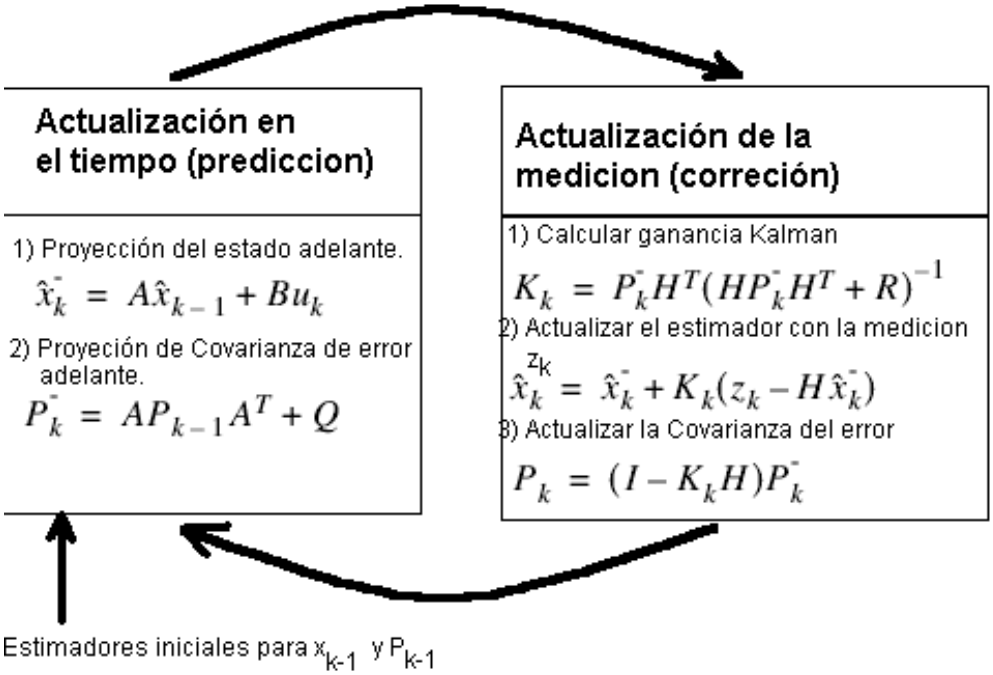


Figura 5-1.- Diagrama de bloques del filtro de Kalman.

La traducción a código de este algoritmo es relativamente simple, para la precisión que aporta. Este código viene implementado en forma de librería bajo el nombre de Kalman.h [12]. Dicha librería está recogida en el Anexo "Código".

5.2.2 Regulador PID

Las siglas PID significan *Proportional Integral Derivative* (Proporcional Integral Derivativo). Es un algoritmo de control de lazo cerrado que consigue una señal de control para llevar el estado del sistema al valor deseado [13]. En el caso del drone, consigue una señal para los motores que llevan a cada ángulo (Yaw, Pitch y Roll) al valor deseado, que el piloto introduce.

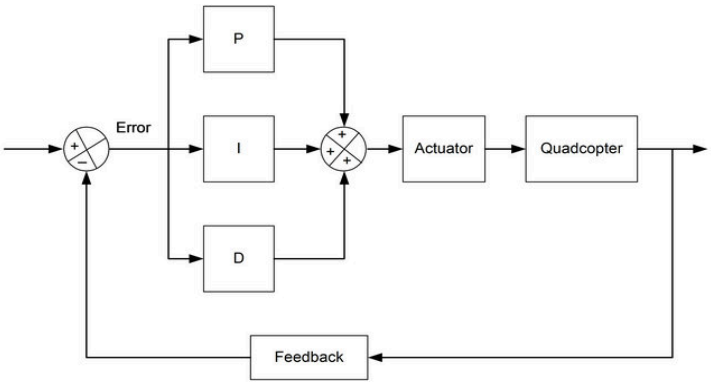


Figura 5-2.- Diagrama de bloques de un regulador PID

A la diferencia entre el valor deseado y el real se le llama error actual, siendo el valor deseado el que el usuario introduce mediante el control del drone; y el valor real, el medido por los sensores (ver Figura 5-2).

Cada parámetro (P, I y D) se emplea en una corrección determinada: el parámetro P actúa sobre el error actual. El parámetro I, sobre la acumulación de error del pasado. El parámetro D, actúa sobre la predicción del error futuro. En cada iteración, se aplican estas tres correcciones para obtener los siguientes valores de las señales de control para los motores, que corrigen el error. Cada parámetro del PID supone el peso de la correspondiente corrección sobre el resultado final. Para una buena estabilización es necesario dar con unos adecuados parámetros del regulador, los cuales son únicos para cada sistema (drone), ya que dependen del peso, el momento de inercia, la rigidez de la estructura, y de una serie de parámetros que son difíciles de controlar como para estandarizar unos valores de P, I y D.

El efecto de cada parámetro sobre la estabilidad del drone es el siguiente:

- Parámetro P: Es el parámetro fundamental para la estabilización. Sin los demás parámetros, un multi-cóptero puede volar relativamente estable. Éste, representa directamente la intensidad con la que se corrige ante la presencia del error. Si se toma un valor bajo, el drone hará correcciones demasiado pequeñas, con lo que no será capaz de mantener la estabilidad cuando el drone comience a torcerse. Para valores demasiado altos, las correcciones son innecesariamente altas, haciendo que el cuadricóptero sobre-oscile.
- Parámetro I: Este parámetro aumenta la precisión de la posición angular. Produce un "efecto memoria" en el error, aumentando la corrección si el error se mantiene demasiado tiempo. Tras ser compensado el error, el término integral sigue "tirando" en el mismo sentido, por lo que, un valor excesivamente alto contrarrestará el efecto del término proporcional, restando estabilidad al sistema.
- Parámetro D: Aumenta la velocidad del drone. Este término amplifica la acción sobre los motores cuando este se mueve con rapidez, lo que se

traduce en una estabilización más rápida, pero, si los otros parámetros no son buenos, este término empeora la estabilización.

Es bastante complicado encontrar los parámetros óptimos para un drone de forma teórica, por lo que se realiza una calibración experimental, observando las reacciones del sistema ante unos parámetros iniciales, y cambiando los valores hasta encontrar una respuesta óptima.

En [14] puede verse un vídeo en el que se muestra el comportamiento de un drone ante distintos valores del PID.

5.3 Desarrollo del código

Una vez introducidas las herramientas necesarias para el tratado de los datos de los sensores, y para la obtención de unas señales de control para los motores, se explica a continuación el funcionamiento de cada una de las partes del código de vuelo. Aquí sólo se explica la lógica del código, estando éste recogido completamente en el Anexo I.

5.3.1 Lectura de los sensores

Como se ha dicho antes, los datos proporcionados por los sensores de Wii han sido descifrados por usuarios, para realizar proyectos, como el desarrollo de IMUs para drones. La lectura se realiza por el protocolo I^2C, el cual se puede utilizar con la librería Wire.h de Arduino. Los sensores responden a determinados comandos, devolviendo los datos en forma de bytes que deben interpretarse mediante un algoritmo determinado. Las funciones necesarias se recogen en la librería, implementada para este proyecto, bajo el nombre de WiiSensors.h. Esta librería recoge información sobre el acceso a los sensores de Wii, recopilada en diferentes sitios web [15].

La librería implementa las siguientes funciones:

- wmpOn(): Envía el comando de activación al giróscopo del Wii Motion Plus (WMP en adelante).
- wmpOff(): Desactiva el WMP.
- wmpSendZero(): Envía un cero al WMP.
- calibrateZeroes(): Obtiene las primeras lecturas del WMP para calibrar a cero las siguientes.
- receiveData(): Realiza las lecturas del WMP.
- receiveRaw(): Recibe lecturas sin calibrar a cero del WMP. (Es lo mismo que llamar a receiveData() sin haber llamado a calibrateZeroes()).
- switchToWmp(): Hace que Arduino escuche al WMP.
- switchToNunchuck(): Hace que Arduino escuche al Nunchuck.
- nunchuck_init(): Activa el Nunchuck.
- send_zero(): Envía un cero al Nunchuck.

- receiveNunchuckData(): Recibe las lecturas del Nunchuck y las decodifica llamando a make_nunchuck_data().
- make_nunchuck_data(): Genera los datos del Nunchuck llamando a nunchuck_decode_byte().
- nunchuck_decode_byte(char): Decodifica cada byte del Nunchuck para quemake_nunchuck_data() los interprete correctamente.

Con estas funciones se pueden obtener los valores de aceleración lineal y angular del sistema, los cuales han de ser procesados mediante una fusión sensorial que resuelve el filtro de Kalman.

5.3.2 Filtrado de los datos

El filtro utilizado es el de Kalman. Como se ha descrito antes, este filtro fusiona las lecturas del acelerómetro y del giróscopo para obtener medidas precisas de los ángulos del drone. Para esto, es necesario conocer los sentidos de los ejes de los sensores y hacer unas pequeñas operaciones previas.

En la Figura 5-3, se muestra la orientación de los sensores respecto al chasis, y el sentido de giro de los ángulos

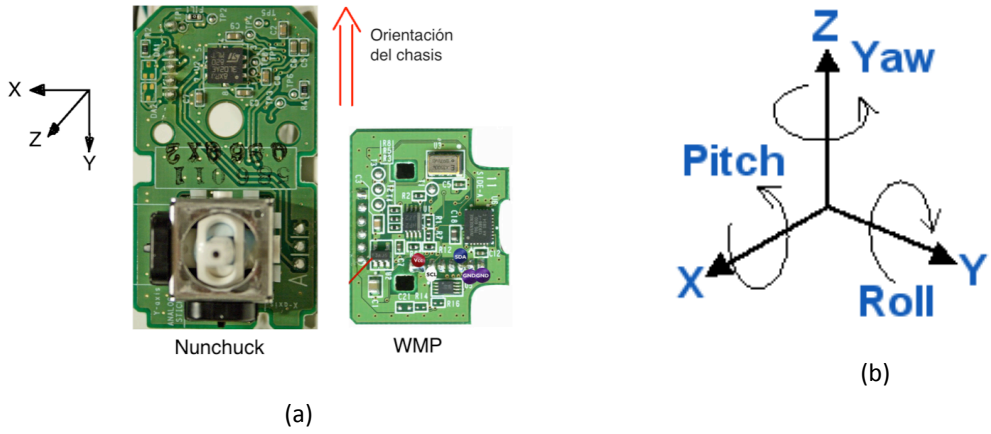


Figura 5-3.- Sentido de los ejes. (a) Orientación de las placas. (b) Sentido de los ángulos.

El procedimiento de medida es el siguiente:

1. Se lee el acelerómetro en los tres ejes.
2. Se pre-calculan los ángulos Pitch y Roll, calculando la arco-tangente de las aceleraciones –Y/Z y –X/Z, respectivamente.
3. Se lee el giróscopo en los tres ejes.
4. Se calcula la velocidad angular, multiplicando por el tiempo transcurrido desde la última medida.
5. Se introducen el pre-ángulo y la velocidad en el filtro de Kalman.

Con estos cinco pasos, se obtienen medidas precisas de los ángulos Pitch y Roll. El ángulo Yaw no puede ser obtenido de esta forma, ya que al variar éste, no cambia la aceleración de la gravedad en ningún eje del acelerómetro, y no puede obtenerse el pre-ángulo para el filtro de Kalman. Para medir este ángulo, es necesario un magnetómetro.

La librería aporta varias funciones para modificar los parámetros del filtro, como Q (ver Figura 5-1). No obstante, no es necesario cambiar ninguno de estos parámetros, y sólo se utilizan en el código las funciones para realizar el filtrado y para establecer el valor de la primera iteración (punto inicial). Dichas funciones son:

- `doublegetAngle(doublenewAngle, doublenewRate, doubledt):`Devuelve el valor del ángulo filtrado al pasarle el pre-ángulo del acelerómetro `(newAngle)`, la velocidad del girócopo`(newRate)` y el tiempo transcurrido desde el cálculo anterior `(dt)`.
- `voidsetAngle(doublenewAngle):` Establece como ángulo inicial el introducido `(newAngle)`.

Para actuar sobre cada eje por separado, se crean dos objetos, uno para el Pitch y otro para el Roll. Cada uno puede ejecutar sus funciones de manera independiente:

```
KalmankalmanPitch;          kalmanPitch.getAngle(…);

KalmankalmanRoll;           kalmanRoll.getAngle(…);
```

Una vez conocida la posición angular del drone, sólo hay que obtener las señales de control para los motores, reguladas por el PID.

5.3.3 Regulador PID

El funcionamiento del PID no es complicado. Simplemente se introducen los valores real y deseado de cada ángulo (medido por los sensores y establecido por el piloto, respectivamente), y se obtiene una salida regulada, en función de los parámetros P, I y D.

Se dispone de una librería que implementa el regulador, llamada PID_v1.h [16]. Esta librería incluye diferentes funciones para realizar la regulación de la señal de control. Concretamente, obtiene los datos a través de punteros[2], para no tener que pasarle los valores a la función en cada cálculo. Utiliza la siguiente sintaxis:

- Parámetros P, I y D: `kp`, `ki` y `kd` respectivamente.
- Ángulo real (sensores): input.
- Ángulo deseado (control del usuario): setpoint.
- Señal de salida: output.
- Tiempo de computación: SampleTime (por defecto, 100 ms).
- Valores mínimo y máximo de la salida: `outMin` y `outMax`.

[2]Puntero: Es un tipo de variable que almacena una dirección de memoria. Se puede leer y escribir en dicha dirección con la sintaxis: &nombreDelPuntero.

La librería implementa, además de la función de cálculo, otras funciones que permiten modificar los diferentes parámetros del regulador (kp, ki, kd, SampleTime, etc.). A continuación se describen las principales funciones de la librería:

- `Compute()`: Ésta es la función principal. Es la que ejecuta el algoritmo de lazo cerrado para la regulación. No devuelve ningún valor directamente, ni es necesario pasarle los valores de los ángulos, ya que funciona con punteros leyendo las direcciones input y setpoint, y escribiendo en la dirección output el resultado de los cálculos, que se calcula como:

```
output=kp* error +ITerm-kd*dInput;
      con:  error = Input – Setpoint
            Iterm = Σ(ki * error)
            dInput = Input - InputAnterior
```

- `Initialize()`: Inicializa los términos I y D, ya que estos dependen de valores anteriores.
- `SetTunings(doubleKp, double Ki, doubleKd)`: Modifica los parámetros del bucle de control (kp, ki, kd).
- `SetSampleTime(intNewSampleTime)`: Modifica el parámetro `SampleTime`.
- `SetOutputLimits(double Min, double Max)`: Modifica los parámetros outMin y outMax.

En este caso, también es necesario crear varios objetos, uno para cada eje controlado. Al llamar a la función Compute() se almacenan en los punteros de salida los valores de las señales de control.

5.3.4 Movimiento de los motores

Esta parte ya ha sido comentada cuando se hablaba de los motores y su utilización. Cada motor opera por medio de señales PWM. Se declaran los pines de cada motor y se escribe el valor deseado, desde 0 a 255. Para el control de los diferentes movimientos del drone, se fija un valor de empuje (throttle) y se suman y se restan los valores de salida de los PID. Aquí es donde se tiene en cuenta la configuración del drone (quad +, quad x, hexa +, hexa x, etc.). Para la configuración del drone de este trabajo, quad x, el código queda:

Figura 5-4.- Distintas configuraciones de cuadricóptero. (a) Quad x. (b) Quad +.

```
#define motorFL  5      //Se asocia cada motor a un pin PWM
#define motorFR  6
#define motorBL  9
#define motorBR  10
//Se escribe el los motores el empuje, y las variaciones
  indicadas por el PID
throttleFL = throttle - OutPitch + OutRoll - OutYaw;
throttleFR = throttle - OutPitch - OutRoll + OutYaw;
throttleBL = throttle + OutPitch + OutRoll + OutYaw;
throttleBR = throttle + OutPitch - OutRoll - OutYaw;
analogWrite(motorFL, throttleFL);
analogWrite(motorFR, throttleFR);
analogWrite(motorBL, throttleBL);
analogWrite(motorBR, throttleBR);
```

Con este código, el drone ya es capaz de mantenerse estable, (siempre y cuando se fijen unos valores adecuados del PID). Sólo falta implementar el control del usuario, mediante el cual se puedan modificar de forma rápida y eficaz los valores de los setpoint, para conseguir los movimientos deseados.

5.3.5 Control del usuario

Como se ha establecido en capítulos anteriores, el control se realiza a través de un módulo Bluetooth. Este tipo de comunicación no ofrece mucho alcance, pero es rápido para vuelos a corta distancia.

La comunicación por Bluetooth permite crear un puerto serie a través del cual se intercambian los datos. La forma más simple de crear un control es, por tanto, escribir en dicho puerto serie los valores de setpoint deseados y que el programa los introduzca en las variables correspondientes. Este método, aunque válido y fácil de implementar, no ofrecería una buena experiencia de vuelo, ya que escribir los valores es algo muy lento, en comparación con las reacciones del drone. Es por esto que se necesita una interfaz gráfica, que ofrezca suficientes botones como para tener control sobre los tres ángulos de movimiento del cuadricóptero.

Existen aplicaciones para Smartphone que permiten enviar comandos cualesquiera por Bluetooth. Un ejemplo es la aplicación para Android: Bluetooth Serial Controller N7 [17], desarrollada por NEXT PROTOTYPES. Esta App permite definir los botones que el usuario quiera, asociarles un comando y enviarlo por Bluetooth.

La interfaz de esta App puede verse en la Figura 5-5.

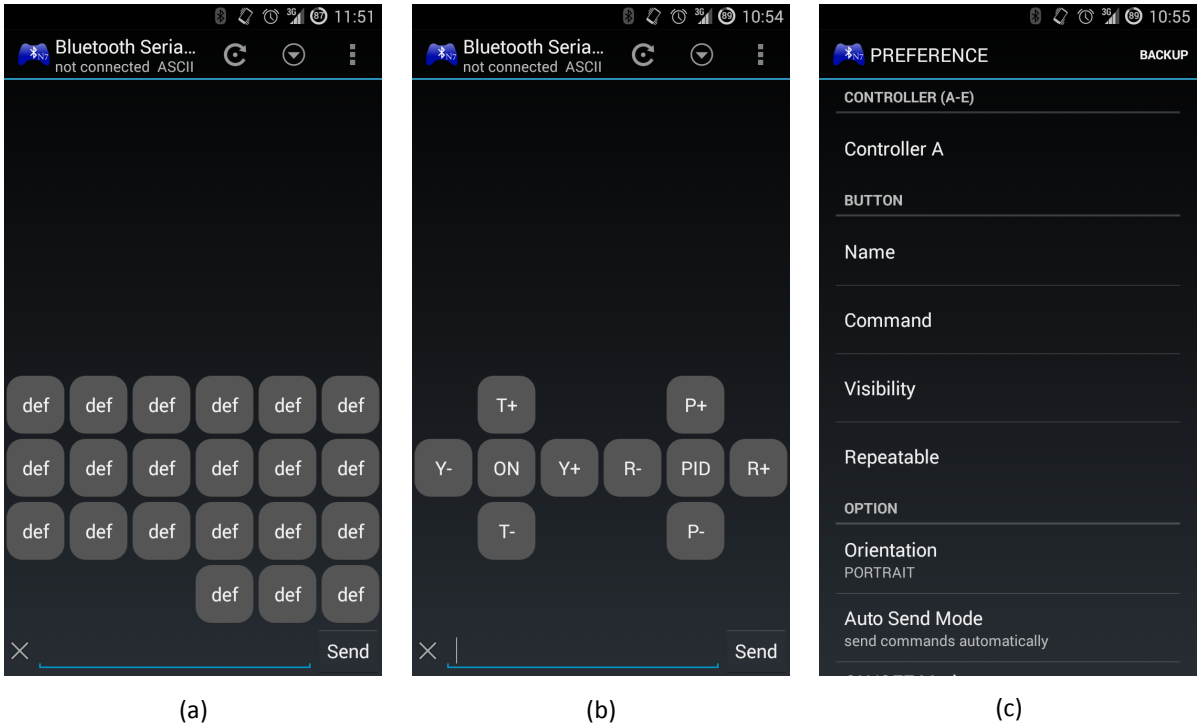


Figura 5-5.- Interfaz de Bluetooth Serial Controller N7. (a) Sin modificar. (b) Definidos los botones y sus comandos. (c) Menú de ajustes.

Los comandos asociados a cada botón de la interfaz, son números y letras por orden (1, 2, …, 9, a, b, …). El software desarrollado de Arduino debe interpretar el valor leído en el puerto serie con un `switch(valor)`, realizando la acción correspondiente en cada `case`.

Se fijan incrementos y decrementos unitarios en los botones T+ y T- para el throttle, y en los botones Y, P y R para el setYaw, setPitch y setRoll, respectivamente. El botón ON activa y desactiva el drone, bloqueando las acciones de los otros botones. El botón PID inicia un menú de configuración de los parámetros del PID, para modificar éstos sin necesidad de apagar el drone y conectarlo al ordenador. Dicho menú opera de la siguiente forma:

1. Tras pulsar el botón PID se interrumpe el loop de Arduino (dentro de un bucle que termina con acciones del usuario), e inicia el menú la configuración del PID, siempre y cuando el drone esté desactivado (mediante el botón ON).
2. Se activa la modificación de los parámetros $\mathbf{k_p}$, de cada eje. Con los botones Y+, P+ y R+, el usuario puede elegir Yaw, Pitch o Roll, respectivamente, para cambiar su k_p con los botones T+ y T-.
3. Al pulsar de nuevo el botón PID, se pasa al parámetro ki del eje que se elija, de la misma manera que en el paso anterior.
4. Igual para pasar al parámetro k_d.

5. Pulsando otra vez PID, se sale del menú y continúa el loop normalmente.

Este menú, agiliza enormemente la búsqueda de los parámetros del PID.

5.3.6 Recopilación en el sketch de Arduino

Todas las librerías descritas anteriormente, y las funciones, ejecutadas en el orden descrito, se recopilan en el cuerpo del programa, un sketch de Arduino llamado Cuadricoptero3D.ino.

Aunque el código completo se recoja en el Anexo I, se describe aquí su estructura.

Un sketch de Arduino debe tener, además de las definiciones de variables y librerías, dos funciones principales: setup y loop. La primera se ejecuta una sola vez al encenderse la placa de Arduino. La segunda se ejecuta una y otra vez, sin parar, hasta que la placa se apague. Por tanto, en setup se programan las tareas de inicialización de los motores y sensores, y las correspondientes calibraciones (por ejemplo, la utilización de la función calibrateZeroes()); y en el método loop se implementa la lectura de sensores e inputs del usuario, y el control de los motores.

Más detalladamente, el código realiza las siguiente tareas:

Definiciones:

- Llamamiento de las funciones utilizadas.
- Creación de los objetos: motores, filtros de Kalman, PIDs y puerto serie.
- Definición de las variables internas del programa.

setup():

- Comienzo de la comunicación por puerto serie.
- Asociación de los motores a los pines PWM y escritura inicial para activarlos (`analogWrite(pinMotor, 10);`).
- Inicialización de los PIDs.
- Inicialización y calibración de los sensores.

loop()

- Lectura de los controles del usuario.
- Lectura de los sensores.
- Filtro de Kalman.
- Regulación por PID.
- Escritura de los motores.
- Salida de mensajes por pantalla.

Con todo en funcionamiento, pueden comenzar las pruebas para la calibración del PID, las cuales se describen en la siguiente sección.

5.4 Pruebas de calibración

Con el trabajo realizado hasta ahora, se consigue tener un drone que responde a las órdenes del usuario, pero que no es capaz de auto-estabilizarse. Por tanto, no es seguro activar el drone sin que esté debidamente sujeto, ya que su comportamiento es, por ahora, impredecible, y podría desestabilizarse provocando el vuelco del mismo, e incluso haciendo daño a quien esté cerca. Para realizar la calibración de forma segura, se utilizan bancos de pruebas y balancines, que limitan los grados de libertad del sistema, y truncan sus movimientos. Existen bancos comerciales en el mercado, pero la solución adoptada es la de atar el drone en los puntos necesarios para limitar su movimiento en el grado deseado.

5.4.1 Ensayo en un eje

La prueba inicial se ilustra en la Figura 5-6. Como puede observarse, se ata el drone por arriba, por abajo y por los lados. El hilo de abajo no tiene tensión inicial, para permitir al cuadricóptero subir un poco, demostrando que está sustentándose por sí mismo durante la prueba. De esta manera, sólo se permite la rotación alrededor de un eje, simplificando la interpretación de la reacción del sistema.

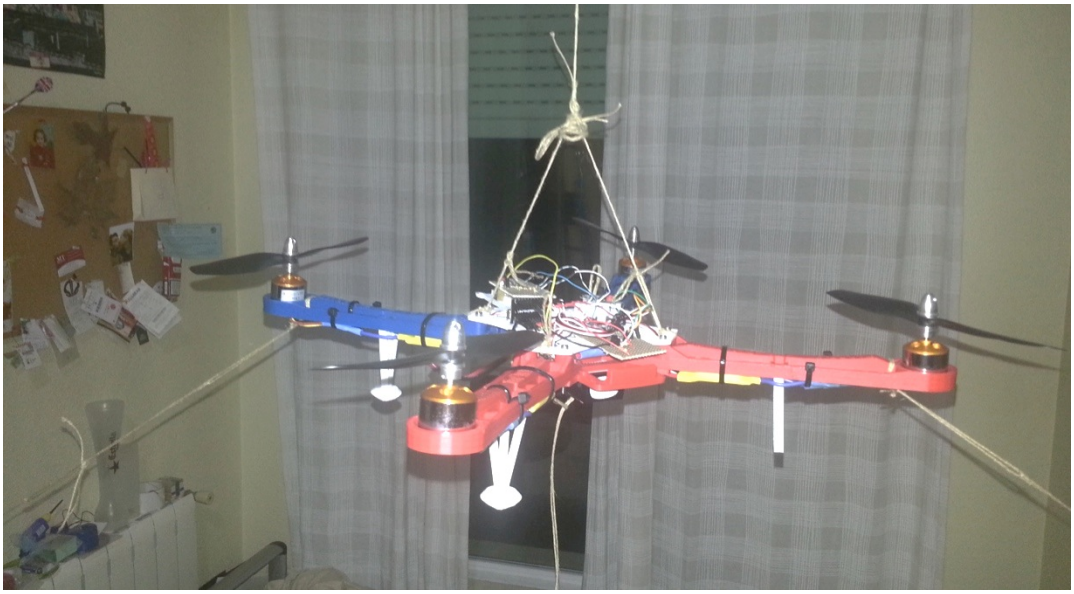


Figura 5-6.- Ensayo en un eje.

Por tratarse de un drone en x, la rotación alrededor de este eje implica una variación equitativa de Pitch y Roll, por lo que, para calibrar el PID, se actúa sobre los parámetros de ambos ejes. Los valores que ofrecen una mejor respuesta en este ensayo son:

- Yaw: P = 1, I = 0, D = 0
- Pitch: P = 1, I = 1, D = 0,25
- Roll: P = 1, I = 1, D = 0,25

5.4.2 *Ensayo en dos ejes*

Se realiza un segundo montaje, que permite al drone inclinarse sobre los dos ejes horizontales (Pitch y Roll), atándolo a una base con ruedas (de un peso mayor que la capacidad de carga del drone), desde sus cuatro brazos (ver Figura 5-7).

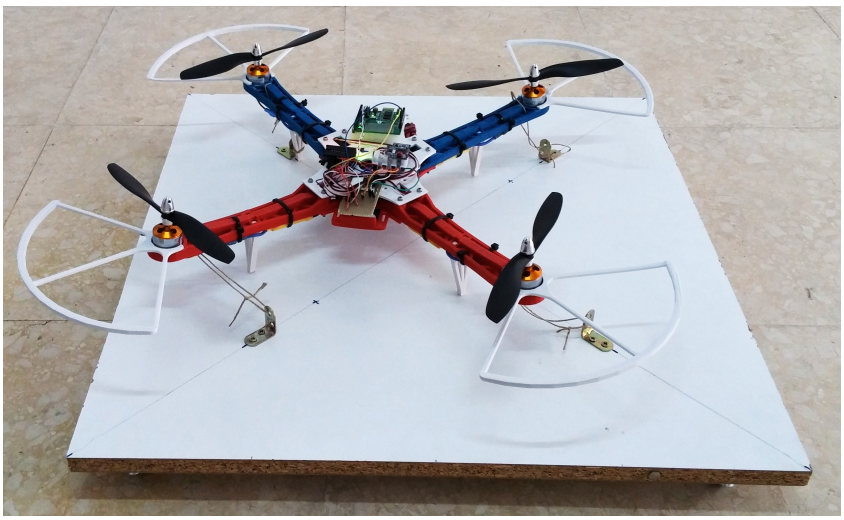


Figura 5-7.- Ensayo en dos ejes.

Este ensayo da más libertad de movimiento al drone, haciendo más realistas sus reacciones, y permitiendo encontrar mejores valores del PID.

En este ensayo los valores que ofrecen una mejor respuesta son:

- Yaw: P = 1, I = 0, D = 0
- Pitch: P = 0,5, I = 0,5, D = 0,25
- Roll: P = 0,5, I = 0,5, D = 0,25

Las diferencias en los valores obtenidos en ambos ensayos se debe a que el sistema reacciona de manera diferente ante los hilos que le impiden ciertos movimientos, los cuales interfieren en la estabilización.

Por tanto, la mejor prueba que se puede hacer es con el drone libre, en el exterior. Así se puede ver su comportamiento real, sin interferencias de los hilos de sujeción. Tras encontrar unos valores aproximados con los ensayos anteriores, se podrá realizar esta última prueba.

Comentar que, en los dos ensayos realizados, se ha fijado el valor de los parámetros del Yaw en (1, 0, 0) para tener sólo control manual sobre el mismo ya que, como se ha comentado antes, no es posible medir de forma eficaz este ángulo sin un magnetómetro.

Capítulo 6

Conclusiones y trabajos futuros

6.1 Conclusiones

En este Trabajo de Fin de Grado se ha diseñado y construido un drone de tipo cuadricóptero, seleccionando los componentes óptimos para la aplicación definida: vuelo teledirigido. Se ha diseñado también un software de vuelo propio utilizando el entorno de programación Arduino IDE. Al haber utilizado una placa Arduino, el sistema es fácilmente escalable y pueden incluirse elementos adicionales, como sensores de proximidad, sensores láser, GPS, barómetro, y un largo etcétera.

La principal conclusión de este trabajo es la enorme diferencia respecto a pocos años atrás: la gran variedad de herramientas y elementos de hardware que existen hoy día, hacen posible lo que hace unos años era impensable. Antes de la existencia de las baterías, sensores y motores de pequeño tamaño actuales, los elementos necesarios para hacer que un aparato vuele era demasiado grandes y pesados para una aplicación como esta.

Como consecuencia, con el hardware de nuestros días se pueden crear estos drones para infinidad de aplicaciones, siendo capaces de aproximar un aparato volador a cualquier punto y a cualquier altura, el único límite es la imaginación.

Por otro lado, gracias a la gran cantidad de información existente en Internet, se pueden integrar nuevas funcionalidades y compartir los resultados con grandes comunidades de usuarios experimentados, formando grandes equipos de trabajo.

6.2 Trabajos futuros

Como continuación y posibles mejoras de este trabajo, se han identificado las siguientes líneas:

Una es el desarrollo de un algoritmo mejorado para el control, por parte del usuario, permitiendo mover el drone a la velocidad deseada. En otras palabras, en lugar de dar más potencia a los motores para alanzar una altura mayor, simplemente mandar la orden de subir a mayor o menor velocidad, siendo el software el responsable de las actuaciones necesarias para alcanzar el comportamiento deseado. Esto significaría subir un nivel en la automatización del sistema.

Otra continuación podría ser la incorporación de cámaras y de software de visión artificial, para conseguir movimientos autónomos.

También puede emprenderse un cambio en el diseño de este trabajo, por otro que permita construir un drone mucho más pequeño, de menor consumo y mayor autonomía, u otro diseño de mayores prestaciones para conseguir drones de mayor capacidad de carga, o incluso sirvan de vehículo.

Como se indica en las conclusiones, las posibilidades son enormes, por lo que hay muchas formas de continuar con un trabajo de este tipo.

Anexo I

Código fuente

En este anexo se incluye todo el código del proyecto, el cual no se ha incluido en los otros capítulos para una mejor organización de la memoria.

1. *Cuerpo del código*

En primer lugar, se incluye el cuerpo del código, el cual ha sido descrito en el Capítulo 5.

```
#include <Wire.h>           //Bibliotecas
#include <WiiSensors.h>
#include <Kalman.h>
#include <PID_v1.h>
#include <SoftwareSerial.h>

SoftwareSerial SerialB(16, 14);     //Se crea el objeto para la comunicación
por puerto serie desde Flyduino

WiiSensors wiiSensors;

Kalman kalmanYaw;
Kalman kalmanPitch;
Kalman kalmanRoll;

int cnt1 = 0, cnt2 = 0, whichVar = 1, whichParam = 1;

double accXangle, accYangle, accZangle;
double YawRate, PitchRate, RollRate, YawAngle;
double Yaw = 0, Pitch = 0, Roll = 0; //Valores de inclinación angular
(Entradas de los PID)
double OutYaw, OutPitch, OutRoll, SetYaw = 0, SetPitch = 144.40, SetRoll =
145.15;  //Valores de Output y Setpoint para los tres PID
PID PIDYaw(&Yaw, &OutYaw, &SetYaw, 1, 0, 0, DIRECT);
PID PIDPitch(&Pitch, &OutPitch, &SetPitch, 0.5, 0.5, 0.25, DIRECT);
```

```
PID PIDRoll(&Roll, &OutRoll, &SetRoll, 0.5, 0.5, 0.25, DIRECT);

uint32_t timer;

#define motorFL  5
#define motorFR  6
#define motorBL  9
#define motorBR  10
int valor;
boolean armed = false, comm = false, tunningPID = false;
double throttle = 10, throttleFL, throttleFR, throttleBL, throttleBR;

void setup()
{
  SerialB.begin(9600);
  pinMode(4, OUTPUT);        //Pin para la alimentación de los sensores
  digitalWrite(4, HIGH);
  pinMode(7, OUTPUT);        //Pin para switchToWmp()
  digitalWrite(7, HIGH);
  pinMode(8, OUTPUT);        //Pin para switchToNunchuck()
  digitalWrite(8, HIGH);

  analogWrite(motorFL, 80);
  analogWrite(motorFR, 80);
  analogWrite(motorBL, 80);
  analogWrite(motorBR, 80);

  SerialB.println("Iniciando...");
  PIDYaw.SetMode(AUTOMATIC);
  PIDYaw.SetSampleTime(50);
  PIDYaw.SetOutputLimits(-40, 40);
  PIDPitch.SetMode(AUTOMATIC);
  PIDPitch.SetSampleTime(50);
  PIDPitch.SetOutputLimits(-40, 40);
  PIDRoll.SetMode(AUTOMATIC);
  PIDRoll.SetSampleTime(50);
  PIDRoll.SetOutputLimits(-40, 40);

  Wire.begin();        // Inicializa la comunicación en la dirección 0x52

  CalibrarSensores()
}

void loop()
{
  LeerControles();    //Lee los controles por el puerto serie
  if(comm == false)   //Comprueba que el control Bluetooth está conectado
  {
    cnt2++;
    if(cnt2 >= 100)   //Si se mantiene desconectado durante 100 ciclos o mas
    {                 //Desactiva el drone
      armed = false;
      throttle = 80;
      analogWrite(motorFL, 80);
      analogWrite(motorFR, 80);
      analogWrite(motorBL, 80);
      analogWrite(motorBR, 80);
      SerialB.println("Se ha perdido la conexion");
      while(!comm)    //Y se queda esperando a que se reconecte
      {
        LeerControles();
      }
      cnt2 = 0;
    }
  }
  else
```

```
        {
          cnt2 = 0;
        }
        //Lectura de los angulos
        LeerSensores();

        PIDYaw.Compute();          //Reguladores PID
        PIDPitch.Compute();
        PIDRoll.Compute();
        //Calculo de las señales de control de los motores
        throttleFL = throttle - OutPitch + OutRoll - SetYaw;
        throttleFR = throttle - OutPitch - OutRoll + SetYaw;
        throttleBL = throttle + OutPitch + OutRoll + SetYaw;
        throttleBR = throttle + OutPitch - OutRoll - SetYaw;
        if(throttle > 100)
        {
          constrain(throttleFL, 100, 254);
          constrain(throttleFR, 100, 254);
          constrain(throttleBL, 100, 254);
          constrain(throttleBR, 100, 254);
          analogWrite(motorFL, throttleFL);
          analogWrite(motorFR, throttleFR);
          analogWrite(motorBL, throttleBL);
          analogWrite(motorBR, throttleBR);
        }
        else
        {
          analogWrite(motorFL, throttle);
          analogWrite(motorFR, throttle);
          analogWrite(motorBL, throttle);
          analogWrite(motorBR, throttle);
        }

        Mensajes();

        cnt1++;
        comm = false;
        delay(1);
      }

      void LeerSensores()
      {
          wiiSensors.switchToWmp();
          wiiSensors.receiveData();
          wiiSensors.switchToNunchuck();
          wiiSensors.receiveNunchuckData();
          //accZangle = (atan2(-wiiSensors.accel_x_axis, -wiiSensors.accel_y_axis) +
PI) * RAD_TO_DEG;
          accXangle  =  (atan2(-wiiSensors.accel_y_axis,  wiiSensors.accel_z_axis)  +
PI) * RAD_TO_DEG;
          accYangle  =  (atan2(-wiiSensors.accel_x_axis,  wiiSensors.accel_z_axis)  +
PI) * RAD_TO_DEG;
          //YawRate = wiiSensors.yaw / 131.0;
          PitchRate = wiiSensors.pitch / 131.0;
          RollRate = wiiSensors.roll / 131.0;
          //Yaw = kalmanYaw.getAngle(accZangle, YawRate, (double)(micros() - timer)
/ 1000000);
          Pitch  =  kalmanPitch.getAngle(accXangle,  PitchRate,  (double)(micros()  -
timer) / 1000000);
          Roll = kalmanRoll.getAngle(accYangle, RollRate, (double)(micros() - timer)
/ 1000000);
          timer = micros();
      }

      void CalibrarSensores()
      {
```

```
        wiiSensors.switchToWmp();  //escucha al WMP
        wiiSensors.wmpOn(); //enciende el WMP
        SerialB.println("Calibrando...");
        delay(100);
        wiiSensors.calibrateZeroes(); //calibra los ceros del WMP
        wiiSensors.switchToNunchuck();
        wiiSensors.nunchuck_init (); //inicializa el Nunchuck
        //Calcula los primeros angulos
        //accZangle = (atan2(-wiiSensors.accel_x_axis, -wiiSensors.accel_y_axis) +
PI) * RAD_TO_DEG;
        accXangle = (atan2(-wiiSensors.accel_y_axis, wiiSensors.accel_z_axis) + PI)
* RAD_TO_DEG;
        accYangle = (atan2(-wiiSensors.accel_x_axis, wiiSensors.accel_z_axis) + PI)
* RAD_TO_DEG;
        //kalmanYaw.setAngle(accZangle);  //Establece los angulos para el filtro de
Kalman
        kalmanPitch.setAngle(accXangle);
        kalmanRoll.setAngle(accYangle);

        for(int i = 0; i < 1000; i++)     //Establece el punto neutro de los setpoint
        {
          LeerSensores();
        }
        //SetYaw = Yaw;
        SetPitch = Pitch;
        SetRoll = Roll;
        SerialB.println("Calibracion terminada");
        timer = micros();
      }
```

La función LeerControles() es un largo switch que engloba todas las acciones de cada uno de los botones de la interfaz de control. Es más largo que el resto del código entero y, en definitiva, no es más que un throttle++; o un setPitch--; en los botones correspondientes. Lo único a remarcar de esta función, es el botón "ON", el cual activa y desactiva el dron y vuelve a hacer la calibración de los sensores. A continuación se muestra el código de este botón:

```
      case 5:  //ON/OFF. El comando que envia este boton es 5
            if(armed)
            {
              armed = false;
              throttle = 80;              //Fija el throttle a 80 (valor minimo)
              analogWrite(motorFL, 80);   //Para los motores
              analogWrite(motorFR, 80);
              analogWrite(motorBL, 80);
              analogWrite(motorBR, 80);
              SerialB.println("Dron desactivado");
              delay(2000);               //Espera a que los motores se paren
              digitalWrite(4, LOW);     //Apaga los sensores
              delay(100);               //Espera 0.1 seg
              digitalWrite(4, HIGH);     //Reactiva los sensores
              CalibrarSensores();
              PIDYaw.SetMode(AUTOMATIC);  //Reinicia los PIDs
              PIDPitch.SetMode(AUTOMATIC);
              PIDRoll.SetMode(AUTOMATIC);
            }
            else
            {
              throttle = 100;
              armed = true;
              SerialB.println("Dron activado");
            }
```

```
        break;
```

La otra función que no se ha incluido es Mensajes(), la cual simplemente muestra por pantalla (en la interfaz de la aplicación móvil) los datos que se van calculando (Yaw, Pitch y Roll; Output de los PIDs, etc.).

2. *Biblioteca WiiSensors.h*

A continuación se presenta el código fuente de la biblioteca WiiSensors. Se muestra el contenido del archivo WiiSensors.cpp, con el código de las funciones. El otro archivo, WiiSensors.h, no se incluye, ya que sólo contiene las definiciones de las funciones del .cpp, y no es necesario para la comprensión del código.

```
#if ARDUINO >= 100
  #include "Arduino.h"
#else
  #include "WProgram.h"
#endif
#include <WiiSensors.h>
#include <Wire.h>
#include <string.h>

#undef int
#include <stdio.h>

byte data[6]; //seis bytes de datos
uint8_t outbuf[6];        // array para almacenar la salida de Arduino
int cnt = 0;

WiiSensors::WiiSensors()     //Constructor de la clase, vacío
{

}

void WiiSensors::wmpOn()
{
  Wire.beginTransmission(0x53); //WMP comienza desactivado en la dirección
0x53
  Wire.write(0xfe); //envia un 0x04 a la dirección 0xFE para activar el WMP
  Wire.write(0x04);
  Wire.endTransmission(); //WMP salta a la dirección 0x52 y se activa
}

void WiiSensors::wmpOff()
{
  Wire.beginTransmission(82);
  Wire.write(0xf0);   //dirección, y después
  Wire.write(0x55);   //comando
  //Wire.write(0x00);
  //Wire.write(0xfb);
  Wire.endTransmission();
}

void WiiSensors::wmpSendZero()
{
  Wire.beginTransmission(0x52); //ahora en la dirección 0x52
  Wire.write(0x00); //envía un cero para indicar que se quiere información
  Wire.endTransmission();
}
```

```
void WiiSensors::calibrateZeroes()
{
  for (int i = 0; i < 10; i++)
  {
    wmpSendZero();
    Wire.requestFrom(0x52,6);
    for (int i = 0; i < 6; i++)
    {
      data[i] = Wire.read();
    }
    yaw0 += (((data[3] >> 2) << 8) + data[0]) / 10; //media de 10 medidas
    pitch0 += (((data[4] >> 2) << 8) + data[1]) / 10;
    roll0 += (((data[5]>>2) << 8) + data[2]) / 10;
  }
  Serial.print("Yaw0:");
  Serial.print(yaw0);
  Serial.print(" Pitch0:");
  Serial.print(pitch0);
  Serial.print(" Roll0:");
  Serial.println(roll0);
}

void WiiSensors::receiveData()
{
  wmpSendZero(); //se envía un cero antes de cada demanda de datos (igual que
el Nunchuck)
  Wire.requestFrom(0x52, 6); //pide los seis bytes del WMP
  for (int i=0;i < 6;i++)
  {
    data[i] = Wire.read();
  }
  yaw = ((data[3] >> 2) << 8) + data[0] - yaw0;
  pitch = ((data[4] >> 2) << 8) + data[1] - pitch0;
  roll = ((data[5] >> 2) << 8) + data[2] - roll0;
}

void WiiSensors::receiveRaw()
{
  wmpSendZero();//se envía un cero antes de cada demanda de datos (igual que
el Nunchuck)
  Wire.requestFrom(0x52,6);  //pide los seis bytes del WMP
  for(int i = 0; i < 6; i++)
  {
    data[i] = Wire.read();
  }
  yaw = ((data[3] >> 2) << 8) + data[0];
  pitch = ((data[4] >> 2) << 8) + data[1];
  roll = ((data[5] >> 2) << 8) + data[2];
}

void WiiSensors::switchToWmp()
{
  digitalWrite(7, LOW);
  digitalWrite(8, LOW);
  digitalWrite(7, HIGH);
}

void WiiSensors::switchToNunchuck()
{
  digitalWrite(7, LOW);
  digitalWrite(8, LOW);
  digitalWrite(8, HIGH);
}

void WiiSensors::nunchuck_init()
{
```

```
  Wire.beginTransmission (0x52);    // transmite al dispositivo 0x52
  Wire.write (0x40);        // envía la direccion de memoria
  Wire.write (0x00);        // envía un cero
  Wire.endTransmission ();    // detiene la transmision
}

void WiiSensors::send_zero()
{
  Wire.beginTransmission(0x52);    // transmite al dispositivo 0x52
  Wire.write(0x00);        // envía un byte
  Wire.endTransmission();    // detiene la transmision
}

void WiiSensors::receiveNunchuckData()
{
  Wire.requestFrom(0x52, 6);
  for (int i = 0; i < 6; i++)
  {
    data[i] = Wire.read();
  }
  make_nunchuck_data();
  send_zero();
}

void WiiSensors::make_nunchuck_data()
{
  for(int i = 0; i < 6; i++)
  {
    outbuf[i] = nunchuck_decode_byte(data[i]);
  }
  int joy_x_axis = outbuf[0];
  int joy_y_axis = outbuf[1];
  accel_x_axis = outbuf[2] * 2 * 2;
  accel_y_axis = outbuf[3] * 2 * 2;
  accel_z_axis = outbuf[4] * 2 * 2;
  boolean z_button = 0;
  boolean c_button = 0;

  // byte outbuf[5] contains bits for z and c buttons
  // it also contains the least significant bits for the accelerometer data
  // so we have to check each bit of byte outbuf[5]
  if((outbuf[5] >> 0) & 1)
  {
    z_button = 1;
  }
  if((outbuf[5] >> 1) & 1)
  {
    c_button = 1;
  }
  if((outbuf[5] >> 2) & 1)
  {
    accel_x_axis += 2;
  }
  if((outbuf[5] >> 3) & 1)
  {
    accel_x_axis += 1;
  }
  if((outbuf[5] >> 4) & 1)
  {
    accel_y_axis += 2;
  }
  if((outbuf[5] >> 5) & 1)
  {
    accel_y_axis += 1;
  }
  if((outbuf[5] >> 6) & 1)
```

```
  {
    accel_z_axis += 2;
  }
  if((outbuf[5] >> 7) & 1)
  {
    accel_z_axis += 1;
  }
}

char WiiSensors::nunchuck_decode_byte(char x)
{
  x = (x ^ 0x17) + 0x17;
  return x;
}
```

3. Biblioteca Kalman.h

La biblioteca Kalman.h [12], desarrollada por Kristian Lauszus, de TKJ Electronics, es la siguiente. Consta sólo del archivo de extensión .h, el cual ya contiene todo el código de la librería.

```
/* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights reserved.

 This software may be distributed and modified under the terms of the GNU
 General Public License version 2 (GPL2) as published by the Free Software
 Foundation and appearing in the file GPL2.TXT included in the packaging of
 this file. Please note that GPL2 Section 2[b] requires that all works based
 on this software must also be made publicly available under the terms of
 the GPL2 ("Copyleft").

 Contact information
 -------------------

 Kristian Lauszus, TKJ Electronics
 Web      :  http://www.tkjelectronics.com
 e-mail   :  kristianl@tkjelectronics.com
 */

#ifndef _Kalman_h
#define _Kalman_h

class Kalman {
public:
    Kalman() {
        /* We will set the variables like so, these can also be tuned by the
user */
        Q_angle = 0.001;
        Q_bias = 0.003;
        R_measure = 0.03;

        angle = 0; // Reset the angle
        bias = 0; // Reset bias

        P[0][0] = 0; /* Since we assume that the bias is 0 and we know the
starting angle (use setAngle), the error covariance matrix is set like so -
see:
http://en.wikipedia.org/wiki/Kalman_filter#Example_application.2C_technical
*/
        P[0][1] = 0;
        P[1][0] = 0;
        P[1][1] = 0;
    };
```

```
            // The angle should be in degrees and the rate should be in degrees per
        second and the delta time in seconds
            double getAngle(double newAngle, double newRate, double dt) {
                //   KasBot   V2      -      Kalman   filter   module   -   http://www.x-
firm.com/?page_id=145
                // Modified by Kristian Lauszus
                // See my blog post for more information:
        http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-
and-how-to-implement-it

                // Discrete Kalman filter time update equations - Time Update
        ("Predict")
                // Update xhat - Project the state ahead
                /* Step 1 */
                rate = newRate - bias;
                angle += dt * rate;

                // Update estimation error covariance - Project the error covariance
        ahead
                /* Step 2 */
                P[0][0] += dt * (dt*P[1][1] - P[0][1] - P[1][0] + Q_angle);
                P[0][1] -= dt * P[1][1];
                P[1][0] -= dt * P[1][1];
                P[1][1] += Q_bias * dt;

                // Discrete Kalman filter measurement update equations – Measurement
        Update ("Correct")
                // Calculate Kalman gain - Compute the Kalman gain
                /* Step 4 */
                S = P[0][0] + R_measure;
                /* Step 5 */
                K[0] = P[0][0] / S;
                K[1] = P[1][0] / S;

                // Calculate angle and bias - Update estimate with measurement zk
        (newAngle)
                /* Step 3 */
                y = newAngle - angle;
                /* Step 6 */
                angle += K[0] * y;
                bias += K[1] * y;

                // Calculate estimation error covariance - Update the error covariance
                /* Step 7 */
                P[0][0] -= K[0] * P[0][0];
                P[0][1] -= K[0] * P[0][1];
                P[1][0] -= K[1] * P[0][0];
                P[1][1] -= K[1] * P[0][1];

                return angle;
            };
            void setAngle(double newAngle) { angle = newAngle; }; // Used to set
        angle, this should be set as the starting angle
            double getRate() { return rate; }; // Return the unbiased rate

            /* These are used to tune the Kalman filter */
            void setQangle(double newQ_angle) { Q_angle = newQ_angle; };
            void setQbias(double newQ_bias) { Q_bias = newQ_bias; };
            void setRmeasure(double newR_measure) { R_measure = newR_measure; };

            double getQangle() { return Q_angle; };
            double getQbias() { return Q_bias; };
            double getRmeasure() { return R_measure; };

        private:
            /* Kalman filter variables */
```

```
    double Q_angle; // Process noise variance for the accelerometer
    double Q_bias; // Process noise variance for the gyro bias
    double R_measure; // Measurement noise variance - this is actually the
variance of the measurement noise

    double angle; // The angle calculated by the Kalman filter - part of the
2x1 state vector
    double bias; // The gyro bias calculated by the Kalman filter - part of
the 2x1 state vector
    double rate; // Unbiased rate calculated from the rate and the calculated
bias - you have to call getAngle to update the rate

    double P[2][2]; // Error covariance matrix - This is a 2x2 matrix
    double K[2]; // Kalman gain - This is a 2x1 vector
    double y; // Angle difference
    double S; // Estimate error
};

#endif
```

4. *Biblioteca PID_v1.h*

Por último, la biblioteca PID_v1.h [16], desarrollada por Brett Beauregard, se facilita a continuación. Sólo se añade el archivo PID_v1.cpp, el cual contiene las funciones. El otro archivo de la librería, sólo contiene las definiciones de las funciones que ya aparecen a continuación.

```
/*
 * Arduino PID Library - Version 1.0.1
 * by Brett Beauregard <br3ttb@gmail.com> brettbeauregard.com
 *
 * This Library is licensed under a GPLv3 License
 **********************************************************************/
#if ARDUINO >= 100
  #include "Arduino.h"
#else
  #include "WProgram.h"
#endif

#include <PID_v1.h>

/*Constructor (...)*********************************************************
 *    The parameters specified here are those for for which we can't set up
 *    reliable defaults, so we need to have the user set them.
 ***************************************************************************/
PID::PID(double* Input, double* Output, double* Setpoint,
        double Kp, double Ki, double Kd, int ControllerDirection)
{

    myOutput = Output;
    myInput = Input;
    mySetpoint = Setpoint;
       inAuto = false;

       PID::SetOutputLimits(0, 255);   //default output limit corresponds to

//the arduino pwm limits

    SampleTime = 100;        //default Controller Sample Time is 0.1 seconds

    PID::SetControllerDirection(ControllerDirection);
    PID::SetTunings(Kp, Ki, Kd);
```

```
        lastTime = millis()-SampleTime;
    }


    /* Compute() **********************************************************
     *            This, as they say, is where the magic happens.  this function
                  should be
     *            called
     *            every time "void loop()" executes.  the function will decide for
                  itself
     *            whether a new
     *            pid Output needs to be computed.  returns true when the output
                  is
     *            computed,
     *   false when nothing has been done.
**********************************************************************************/
    bool PID::Compute()
    {
       if(!inAuto) return false;
       unsigned long now = millis();
       unsigned long timeChange = (now - lastTime);
       if(timeChange>=SampleTime)
       {
          /*Compute all the working error variables*/
             double input = *myInput;
          double error = *mySetpoint - input;
          ITerm+= (ki * error);
          if(ITerm > outMax) ITerm= outMax;
          else if(ITerm < outMin) ITerm= outMin;
          double dInput = (input - lastInput);

          /*Compute PID Output*/
          double output = kp * error + ITerm- kd * dInput;

             if(output > outMax) output = outMax;
          else if(output < outMin) output = outMin;
             *myOutput = output;

          /*Remember some variables for next time*/
          lastInput = input;
          lastTime = now;
             return true;
       }
       else return false;
    }


    /* SetTunings(...)*************************************************
     * This function allows the controller's dynamic performance to be adjusted.
     * it's called automatically from the constructor, but tunings can also
     * be adjusted on the fly during normal operation
**********************************************************************************/
    void PID::SetTunings(double Kp, double Ki, double Kd)
    {
       if (Kp<0 || Ki<0 || Kd<0) return;

       dispKp = Kp; dispKi = Ki; dispKd = Kd;

       double SampleTimeInSec = ((double)SampleTime)/1000;
       kp = Kp;
       ki = Ki * SampleTimeInSec;
       kd = Kd / SampleTimeInSec;
```

```
  if(controllerDirection ==REVERSE)
   {
      kp = (0 - kp);
      ki = (0 - ki);
      kd = (0 - kd);
   }
}

/* SetSampleTime(...) *********************************************
 * sets the period, in Milliseconds, at which the calculation is performed
 ******************************************************************/
void PID::SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime
                      / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}

/* SetOutputLimits(...)****************************************************
 *     This function will be used far more often than SetInputLimits.  while
 *  the input to the controller will generally be in the 0-1023 range (which
 *            is
 *            the default already,)  the output will be a little different.
                Maybe they'll
 *            be doing a time window and will need 0-8000 or something.  or
                maybe they'll
 *            want to clamp it from 0-125.  who knows.  at any rate, that can
                all be done here.
 **************************************************************************/
void PID::SetOutputLimits(double Min, double Max)
{
   if(Min >= Max) return;
   outMin = Min;
   outMax = Max;

   if(inAuto)
   {
          if(*myOutput > outMax) *myOutput = outMax;
          else if(*myOutput < outMin) *myOutput = outMin;

          if(ITerm > outMax) ITerm= outMax;
          else if(ITerm < outMin) ITerm= outMin;
   }
}

/* SetMode(...)****************************************************
 * Allows the controller Mode to be set to manual (0) or Automatic (non-zero)
 * when the transition from manual to auto occurs, the controller is
 * automatically initialized
 ******************************************************************/
void PID::SetMode(int Mode)
{
    bool newAuto = (Mode == AUTOMATIC);
    if(newAuto == !inAuto)
    {  /*we just went from manual to auto*/
        PID::Initialize();
    }
    inAuto = newAuto;
}
```

```
/* Initialize()****************************************************
 *     does all the things that need to happen to ensure a bumpless transfer
 *  from manual to automatic mode.
 ******************************************************************/
void PID::Initialize()
{
   ITerm = *myOutput;
   lastInput = *myInput;
   if(ITerm > outMax) ITerm = outMax;
   else if(ITerm < outMin) ITerm = outMin;
}

/* SetControllerDirection(...)*************************************
 * The PID will either be connected to a DIRECT acting process (+Output leads
 * to +Input) or a REVERSE acting process(+Output leads to -Input.)  we need
to
 * know which one, because otherwise we may increase the output when we should
 * be decreasing.  This is called from the constructor.
 ******************************************************************/
void PID::SetControllerDirection(int Direction)
{
   if(inAuto && Direction !=controllerDirection)
   {
         kp = (0 - kp);
      ki = (0 - ki);
      kd = (0 - kd);
   }
   controllerDirection = Direction;
}

/* Status Funcions*************************************************
 * Just because you set the Kp=-1 doesn't mean it actually happened.  these
 * functions query the internal state of the PID.  they're here for display
 * purposes.  this are the functions the PID Front-end uses for example
 ******************************************************************/
double PID::GetKp(){ return  dispKp; }
double PID::GetKi(){ return  dispKi;}
double PID::GetKd(){ return  dispKd;}
int PID::GetMode(){ return  inAuto ? AUTOMATIC : MANUAL;}
int PID::GetDirection(){ return controllerDirection;}
```

Referencias

[1] Cuadricóptero Pelican de AscTec. Disponible on-line en: http://www.asctec.de/uav-applications/research/products/asctec-pelican/

[2] Penn Quadrotors. Disponible on-line en: http://www.upenn.edu/spotlights/penn-quadrotors-ted

[3] Vídeo del Proyecto Penn Quadrotors. Disponible on-line en: https://www.youtube.com/watch?v=4ErEBkj_3PY

[4] Dron PixHawk Cheetah Bravo. Disponible on-line en: https://pixhawk.ethz.ch/micro_air_vehicle/quadrotor/cheetah

[5] Driver ROS para los drones AR. Drone de Parrot. Disponible on-line en: https://github.com/AutonomyLab/ardrone_autonomy

[6] Proyecto STARMAC. Disponible on-line en: http://hybrid.eecs.berkeley.edu/starmac/

[7] Dron desarrollado en la Universidad de Michigan. Disponible on-line en: http://msutoday.msu.edu/news/2013/msu-lands-first-drone/

[8] Dron desarrollado en la Universidad de Zúrich. Disponible on-line en: http://www.dfab.arch.ethz.ch/web/e/forschung/240.html

[9] Dron desarrollado en la Cornell University. Disponible on-line en: http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2012/yk579_jl2782_tnn7/yk579_jl2782_tnn7/intro.html

[10] Explicación detallada del diseño y construcción de un dron "casero". Disponible on-line en: http://blog.deinventos.com/construyendo-un-cuadricoptero-i/

[11] Información sobre el filtro de Kalman. Disponible on-line en: http://www.ie.itcr.ac.cr/gaby/Control_Automatico/Presentaciones/09_ControlconFiltrodeKalman_v12s01.pdf

y en: http://www.cimat.mx/~alram/VC/MSAA.htm

[12] Código de un filtro de Kalman. Disponible on-line en: https://github.com/TKJElectronics/KalmanFilter

[13] Información sobre reguladores PID. Disponible on-line en: http://blog.oscarliang.net/quadcopter-pid-explained-tuning/

[14] Video del efecto de los parámetros del PID en un dron. Disponible on-line en: https://www.youtube.com/watch?v=YNzqTGEl2xQ

[15] Información para el acceso a los sensores de Wii. Disponible on-line en: http://randomhacksofboredom.blogspot.com.es/2009/07/motion-plus-and-nunchuck-together-on.html

y en: http://forum.arduino.cc/index.php/topic,8661.0.html

[16] Librería que implementa un regulador PID. Disponible on-line en: https://github.com/br3ttb/Arduino-PID-Library

[17] App: Bluetooth Serial Controller, desarrollada por NEXT PROTOTYPES. Disponible on-line en Google Play: https://play.google.com/store/apps/details?id=mBluetoothSerialController.Nexus7&hl=es